Performance Preorder and Competitive Equivalence *

Flavio Corradini† Roberto Gorrieri‡ Marco Roccetti‡

†Dipartimento di Scienze dell'Informazione, Università "La Sapienza"
Via Salaria 113, I-00189 Roma, Italy
e-mail: corradini@dsi.uniroma1.it

‡Dipartimento di Scienze dell'Informazione, Università di Bologna
Piazza di Porta S. Donato 5, I-40127 Bologna, Italy
e-mail: {gorrieri,roccetti}@cs.unibo.it

Abstract

A preorder based on execution speed, called performance preorder, is introduced for a simple process algebra with durational actions. Two processes E and F are related -$E \sqsubseteq_p F$- if they have the same functionality (in this case, we have chosen strong bisimulation equivalence) and E is at least as fast as F. Hence, this preorder supports the stepwise refinement "from specification to implementation" by increasing efficiency while retaining the same functionality. We show that the problem of finding faster implementations for a specification is connected to the problem of finding more distributed implementations of the same specification. Both performance preorder and the induced equivalence, called competitive equivalence, are provided with sound and complete axiomatizations for finite agents.

1 Introduction

Recently, in the field of semantics for process description languages, a great deal of interest has been stirred up by equivalence notions which incorporate some measure of efficiency (see, e.g., [20, 4, 2, 14]). Some of them [2, 3, 12, 14] rely on the basic idea that processes are discriminated not only according to their *functionality* (what actions the processes can do), but also considering their *performance* (a measure of the time consumed for their execution).

Particularly relevant for the present work is [14], where two of the authors proposed a theory of processes based on durational actions. The main features of this approach are the following:

- *Actions are durational*: External to the syntax, one has to specify a duration function for actions. This allows us to consider a normal syntax for processes and to include time information only in the semantics: the evolution of a process is represented by a transition system labelled by pairs of the form $\langle a, n\rangle$, meaning that action a has been completed at time n.
- *Actions are urgent* (as opposed to patient or lazy): There is no delay in between two consecutive actions in the same sequential subsystem.[1]
- *Absence of timed transitions*: A change of the state can occur (i.e., a transition is performed) only due to the execution of real "programmable" actions of the process; this is in contrast to most approaches where time passing can cause a state change. Hence, the transition system is not "dense" in our approach, even in case the time domain be so.

*The first author has been partially supported by CNR Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo, Contract n. 93.01599.69 -LAMBRUSCO. The second and third authors have been partially supported by MURST, CNR and Esprit BRA n. 8130 LOMAPS.

[1] A form of busy-waiting is allowed in [14] only for actions involved in a synchronization step; however, as the language we consider here does not include communication, no busy-waiting is allowed.

- *Executions are asynchronous*: Time is not used for imposing a synchronous view to the system; in several approaches, time is represented through a special global action which all the subprocesses must synchronize upon. Here, instead, each subprocess evolves independently of the other ones,[2] as in a real distributed system where clocks are not required to be synchronized.

- *Presence of "well-caused but ill-timed" paths* [2]: In order to obtain a semantics which is comparable with standard interleaving semantics [18], one has to consider observations where time may go backwards; this phenomenon occurs, however, only when the two adjacent transitions are generated by independent events, so that their "well-caused but ill-timed" occurrence does not harm. The "ill-timed" paths show that the "interleaving eye" we use to look at computations may give an order to observations which is different of the order of execution. Moreover, their presence offers a much simpler semantic treatment, based on the asynchrony of execution.

These basic assumptions fit well with the intuition that an observer is simply looking at the system, without interacting with it. In other words, our point of view is that of a (CCS-like) programmer who writes implementations, checks if they are equivalent to their specifications and wants to order them w.r.t. to their execution speed. In many other approaches, instead, the goal is that of defining a "timed" process algebra where the specification of time (and its real-time constraints) is the crucial aspect.

In [14], standard bisimulation [18] on this transition system (with labels of the form $\langle a, n\rangle$) has been called performance equivalence, which equates systems whenever they perform the same actions in the same amount of time. In the present paper we want to study the problem of ordering processes with respect to speed, allowing us to consider degrees of efficiency among processes. This is obtained by introducing a preorder, denoted $\sqsubseteq_p$, based on the following assumptions:

- The preorder must relate *functionally equivalent* processes: It is quite reasonable to require that only processes exhibiting the same functionality are related, as there is no reason to order w.r.t. speed processes doing different things. The choice of the actual functional equivalence to be preserved is free; in this paper we adopt interleaving (strong) bisimulation [18] just because (*i*) it is a well-known and widely accepted equivalence, (*ii*) it is more discriminating than most interleaving equivalences (hence the results we prove for it should hold also for more abstract semantics) and (*iii*) it is algorithmically more tractable than other interleaving semantics.

- The preorder must order processes w.r.t. speed: if $E \sqsubseteq_p F$, then we expect that E to be more efficient in doing the same things F is able to do.

The kernel of this preorder, $\sqsubseteq_p \cap \sqsupseteq_p$, gives rise to a new equivalence called *competitive equivalence* and denoted $\simeq_c$, which relates processes by considering only the optimal (or fastest) and worst (or slowest) executions. This equivalence is more discriminating than standard interleaving bisimulation. Indeed, the equation $a \mid b \ \simeq_c \ a.b + b.a$ does not hold in our setting because $a \mid b \sqsubseteq_p a.b + b.a$ but $a.b + b.a \not\sqsubseteq_p a \mid b$. Competitive equivalence is strictly weaker than performance equivalence and we claim that the former is more appropriate when one is interested in estimating the time intervals for the optimal and worst cases only.

We study these problems on a simple process algebra not dealing with synchronization, restriction and silent moves (hinted only in Section 8). The main reasons are:

- Simple technicalities, almost standard for those working with non-interleaving equivalences, do not obscure our ideas with difficult details.

[2]Only in synchronization steps time plays a role, as a synchronization is possible only if the time when both partners are ready is the same [14].

- Strong results, some of which cannot be easily extended to a richer language (see the conclusions for more details).

- Ease of comparison with other preorders (e.g., the pomset preorder of [1] has been defined on the same language we use).

The paper is organized as follows. In the next section we introduce the simple process algebra we investigate, while its operational semantics is reported in Section 3. Section 4 is devoted to the behavioural semantics; we first recall performance equivalence from [14] and then describe our new proposal of performance preorder and competitive equivalence. Section 5 is devoted to the algebraic characterization of performance preorder (consequently, of competitive equivalence) and of performance equivalence. The two sound and complete axiomatizations differ only with respect to one inference rule which has inequational nature for the former relation while it has equational nature for the latter. In Section 6 we formally compare competitive equivalence and performance equivalence with the best known untimed non-interleaving semantics proposed in the literature. This includes a comparison of our performance preorder with the *pomset preorder* [1] (relating sequentiality and nondeterminism to concurrency without reducing the latter to the former) and with the *location preorder* [6] (relating processes w.r.t. spatial distribution). The latter comparison is particularly interesting as it proves the (quite expected) result that the more distributed a process is the more efficient it is; even more, we show that in some cases our preorder captures more correctly the intuitive idea of distribution (see Proposition 6.3). In Section 7 we present a simple application of the approach to the specification of concurrent systems. Finally, Section 8 is devoted to concluding remarks, to comparison with related work, to extending the approach to communicating systems and to further research.

2 The Language

Let *Act* be the set of *actions*, ranged over by a, b, c, ... and X be the set of variables, ranged over by x, y, z, ... The process terms are freely generated by the following syntax:

$$F \ ::= nil \quad \Big| \quad a.F \quad \Big| \quad F+F \quad \Big| \quad F \mid F \quad \Big| \quad x \quad \Big| \quad \mathrm{rec}\ x.\,F$$

This is the subset of *CCS* processes [18] where the operators of restriction and relabelling are excluded and where the parallel operator $\mid$ does not allow communication.

In the following of this paper, we will focus on the set P of the closed terms (usually called *agents* or *processes*), ranged over by E. The set of *finite processes* (i.e., without recursion) will be denoted by P_{fin}.

3 The Operational Semantics

The final goal of any program is to be executed, and the measure of the time it consumes strictly depends on the chosen machine. As a matter of fact, this elementary form of performance evaluation can be carried out only if we know how much time is taken by the computer to execute each action. Our assumptions on abstract machines are the following:

1) *maximal parallelism:* whenever a new subagent is activated, there is always a processor free, ready to execute it. In other words, there is never the need of serializing parallel computations. In this way, we are investigating the theoretical, optimal performance measure, which does not consider possible constraints due to the possible limited number of available processors.

2) *eagerness:* there is no time-passing in between the execution of actions from the same sub-agent; equivalently, actions happen as soon as possible. [3]

3) *static duration:* the amount of time needed for the execution of a particular action is fixed once and for all on the basis of the features of the abstract machine. As the duration of actions can be different for different machines, we should take this parameter into account in our semantic description. One possibility is to introduce *duration functions* (ranged over by f, g) which associate to each action the positive integer of time units needed for its execution. In the following, we choose any duration function $f : Act \to N^+$, simply to fix this parameter. We make this assumption also for the sake of simplicity, in order to keep time separate from the syntax.

Our language is equipped with an SOS semantics in terms of labelled transition systems.

Definition 3.1 A *labelled transition system* is a triple $\langle S, M, T\rangle$ where S is a set of *states*, M is a set of *labels* and $T = \{\stackrel{\mu}{\longrightarrow} \subseteq S \times S \mid \mu \in M\}$ is the *transition relation*. We will write $s \stackrel{\mu}{\longrightarrow} s'$ instead of $\langle s, s'\rangle \in \stackrel{\mu}{\longrightarrow}$.

In our case, the states are terms of a syntax extending the one for agents with a local *clock prefixing* operator, $n \Rightarrow \_$, which records the evolution of different parts of a distributed state.

Definition 3.2 The states are terms generated by the following syntax:

$$s \ ::= n \Rightarrow nil \ \Big| \ n \Rightarrow a.E \ \Big| \ s + s \ \Big| \ s \mid s \ \Big| \ n \Rightarrow \mathrm{rec}\ x.\, F$$

where E and rec $x.\,F$ denote agents. The set of states is denoted by S.

$$\boxed{\begin{array}{rcl} n \Rightarrow (E \mid E') & = & (n \Rightarrow E) \mid (n \Rightarrow E') \\ n \Rightarrow (E + E') & = & (n \Rightarrow E) + (n \Rightarrow E') \end{array}}$$

Table 1: Clock Distribution Equations.

The equations in Table 1, called *clock distribution equations*, show that a term $n \Rightarrow E$ can be "canonically" reduced to a state, when interpreting these equations as rewrite rules from left to right.

Each transition is labelled by pairs of the form $\langle a, n\rangle$ meaning that action a has been completed exactly n time units after the computation began.

Definition 3.3 The set of labels for the transition system is $\Pi = Act \times N^+$.

The transition relation is given through a set of inference rules, listed in Table 2, defined in a structural inductive manner (apart from the rule for recursion). It is worthwhile observing that these rules are parametric w.r.t. the chosen duration function f. Hence, to be precise, we should write $\to_f$. For the sake of simplicity, the subscript will always be omitted because clear from the context.

[3] These two assumptions (maximal parallelism and eagerness) are not to be confused with the so-called *maximal progress* assumption which is a combination of the above conditions with the requirement that early actions can always disable later conflicting actions. E.g., in a CCS term like $((a.\bar{b}.c|\bar{b}.c)|b)\backslash\{b\}$, both synchronizations on $\bar{b}$ are allowed in the approach of [14], while the synchronization with the $\bar{b}$ not caused by a is forbidden if maximal progress is assumed.

Act	$\dfrac{}{n \Rightarrow a.E \xrightarrow{\langle a,k\rangle} k \Rightarrow E}$	$k = n + f(a)$
Alt_1	$\dfrac{s_1 \xrightarrow{\langle a,n\rangle} s_1'}{s_1 + s_2 \xrightarrow{\langle a,n\rangle} s_1'}$	
Alt_2	$\dfrac{s_2 \xrightarrow{\langle a,n\rangle} s_2'}{s_1 + s_2 \xrightarrow{\langle a,n\rangle} s_2'}$	
Par_1	$\dfrac{s_1 \xrightarrow{\langle a,n\rangle} s_1'}{s_1 \mid s_2 \xrightarrow{\langle a,n\rangle} s_1' \mid s_2}$	
Par_2	$\dfrac{s_2 \xrightarrow{\langle a,n\rangle} s_2'}{s_1 \mid s_2 \xrightarrow{\langle a,n\rangle} s_1 \mid s_2'}$	
Rec	$\dfrac{n \Rightarrow E[\mathrm{rec}\ x.\, E/x] \xrightarrow{\langle a,k\rangle} s'}{n \Rightarrow \mathrm{rec}\ x.\, E \xrightarrow{\langle a,k\rangle} s'}$	

Table 2: The Structural Rules for the Operational Semantics.

A few comments on the operational rules are now in order. The rule for action prefixing states that an action a, executable at time n, is completed at time $k = n + f(a)$; the number k denotes the time which passed for all the sequential subsystems of E. Note that $k \Rightarrow E$ may be not a state; in such a case, applications of the clock distribution equations will eventually transform $k \Rightarrow E$ into a state. Note also that different occurrences of the same action a last the same amount of time, namely $f(a)$, each time a is encountered. Rules Alt_1, Alt_2, Par_1 and Par_2 are as usual. Finally, note that $n \Rightarrow E[\mathrm{rec}\ x.\, E/x]$ in the premise of rule *Rec* may need applications of the clock distribution equations to become a state.

4 Behavioural Semantics

In this section we first recall the definition of performance equivalence from [14] and prove that it is a congruence. Then, we investigate the performance preorder and the induced equivalence, called *competitive*.

4.1 Performance Equivalence

The observational semantics we are interested in is based on the branching-time semantics of bisimulation; as a matter of fact, performance bisimulation is nothing but (ordinary) strong bisimulation on the transition system labelled by pairs of the form $\langle a, n\rangle$.

Definition 4.1 (*Performance Equivalence*)

1) Let *Rel* denote the set of binary relations over S. The functional $R : Rel \to Rel$ is defined, for each $\Re \in Rel$, as follows: $(s_1, s_2) \in R(\Re)$ if, for each $a \in A$,

i) $s_1 \stackrel{\langle a,n\rangle}{\longrightarrow} s'_1$ implies $s_2 \stackrel{\langle a,n\rangle}{\longrightarrow} s'_2$ such that $(s'_1, s'_2) \in \Re$;

ii) $s_2 \stackrel{\langle a,n\rangle}{\longrightarrow} s'_2$ implies $s_1 \stackrel{\langle a,n\rangle}{\longrightarrow} s'_1$ such that $(s'_1, s'_2) \in \Re$.

2) A relation $\Re \in Rel$ will be called an $R - bisimulation$ if $\Re \subseteq R(\Re)$.

3) We say that two states s_1 and s_2 are *f-performance equivalent*, denoted $s_1 \sim^f s_2$, if and only if there exists an $R - bisimulation$ $\Re$ such that $(s_1, s_2) \in \Re$.

4) We say that two agents E_1, E_2 are *f-performance equivalent*, denoted $E_1 \sim_p^f E_2$, if and only if $0 \Rightarrow E_1 \sim^f 0 \Rightarrow E_2$.

Again, when f is clear from the context, we can omit the superscript f, as in $s_1 \sim s_2$ and $E_1 \sim_p E_2$. This observational semantics equates systems whenever they perform the *same* actions at the same time thus introducing a simple form of performance evaluation in process algebra.

Now we want to present a new, alternative formulation of the operational rules. This new formulation is given because it will be helpful

i) in order to prove that performance bisimulation is a congruence for all the operators of the language;

ii) in the definition of the axiomatization of the semantic congruences we are going to investigate, and also

iii) in comparing these performance-based semantics with other non interleaving ones.

The rules of Table 2 define essentially a tyft system [16] for which strong bisimulation equivalence (hence, in this case, also performance equivalence) is a congruence.

The relevant exceptions are rules *Act* and *Rec*, which do not fit in the format because they do not only exploit the top level operator name. In order to define new operational rules fitting the format, we need to extend the algebra of states by including agents directly and by removing the clock distribution equations.

The set T of *extended states* is generated as follows:

$$t \ ::= nil \ \Big| \ a.E \ \Big| \ \mathrm{rec}\ x.\,F \ \Big| \ t + t \ \Big| \ t \,|\, t \ \Big| \ n \Rightarrow t$$

where E and rec $x.\,F$ are agents. The set of finite extended states is denoted T_{fin}.

A new transition relation $\longmapsto \subseteq T \times \Pi \times T$ is defined on extended states. Rules *Act* and *Rec* are replaced by the axiom and inference rules reported in Table 3, which presents an additional rule for the clock prefixing operator. The whole set of operational rules includes also rules Alt'_1, Alt'_2, Par'_1 and Par'_2 which are as their corresponding rules in Table 2 where the transition relation $\longmapsto$ is substituted for $\longrightarrow$.

There is a clear correspondence between states and extended states, which can be made explicit as follows. Let $\equiv$ denote the *structural congruence* induced by the four equations in Table 4. Then it can be proved that

Proposition 4.1 Given an extended state t, there is a unique state s such that $t \equiv s$.

The proof of this proposition is based on the simple observation that if we first use the fourth axiom as a rewrite rule transforming each occurrence in t of a process E into $0 \Rightarrow E$, and then we consider the first three equations as rewrite rules from left to right, hence the so produced term rewriting system is confluent and terminating, with states as normal forms.

Note that the new operational semantics $\longmapsto$ on (the free algebra of) extended states makes sound, w.r.t. strong bisimulation, all the equations in Table 4. This observation suggests that there

Act'	$\dfrac{}{a.E \overset{\langle a, f(a)\rangle}{\longmapsto} f(a) \Rightarrow E}$
Rec'	$\dfrac{E[\mathrm{rec}\ x.\, E/x] \overset{\langle a,n\rangle}{\longmapsto} t}{\mathrm{rec}\ x.\, E \overset{\langle a,n\rangle}{\longmapsto} t}$
Clock	$\dfrac{t \overset{\langle a,k\rangle}{\longmapsto} t'}{n \Rightarrow t \overset{\langle a,n+k\rangle}{\longmapsto} n \Rightarrow t'}$

Table 3: Operational Rules in tyft Format.

$n \Rightarrow (t \mid t')$	$=$	$(n \Rightarrow t) \mid (n \Rightarrow t')$
$n \Rightarrow (t\ +\ t')$	$=$	$(n \Rightarrow t)\ +\ (n \Rightarrow t')$
$n \Rightarrow (m \Rightarrow t)$	$=$	$(n + m) \Rightarrow t$
E	$=$	$0 \Rightarrow E$

Table 4: Equations for Structural Congruence

is another transitional semantics for extended states, which exploits the structural congruence $\equiv$ and the transition relation $\longrightarrow$. The new transition relation is denoted by $\Vdash\!\!\longrightarrow \subseteq T \times \Pi \times T$ and is defined by the following structural rule:

$$\mathit{Struct}\ \ \frac{t_1 \equiv\ s_1 \overset{\langle a,n\rangle}{\longrightarrow} s_2 \equiv\ t_2}{t_1 \overset{\langle a,n\rangle}{\Vdash\!\!\longrightarrow} t_2}$$

The following proposition states that the two transitional semantics on extended states coincide.

Proposition 4.2 Let t_1, t_2 be extended states. Then, $t_1 \overset{\langle a,k\rangle}{\longmapsto} t_2$ if and only if $t_1 \overset{\langle a,k\rangle}{\Vdash\!\!\longrightarrow} t_2$.

This observation should definitely clarify that the operational semantics on extended states, defined by $\longmapsto$, is exactly the same we have proposed for states in Table 2, as all the states in the same congruence class have the same semantics.

Now, a result by [16] ensures that strong bisimulation is a congruence on the set of extended states. In order to prove that this is a congruence also on the subalgebra of states, it is enough to observe that there is no transition from a state to an extended state in the transition system defined by the rules in Table 2, i.e., the transition relation is closed on states. Since bisimulation is a congruence on states (a fortiori when the clock prefixing is 0), then the following proposition follows immediately.

Proposition 4.3 Performance equivalence is a congruence on P.

Despite of its technical convenience (see also Section 5), we have preferred not to stick to this alternative operational formulation from the beginning because it has the drawback that the terms representing the states "grow in size", as long as the computation proceeds: indeed, every occurrence of an action adds another clock prefixing to the term. Moreover, time information are not located in the (unique) clock prefix of the sequential components of an agent (as it is for the original approach), rather they are spread in the term, thus making it harder to understand the updating of the clock values as the computation proceeds.

4.2 The Performance Preorder

In order to relate processes with respect to speed, we propose a preorder $\sqsubseteq_p^f$ over P: $E_1 \sqsubseteq_p^f E_2$ if and only if E_1 and E_2 have the same functional behavior but E_1 is faster than E_2 according to duration function f.

Definition 4.2 (*Performance Preorder*)

1) Let Rel denote the set of binary relations over S. The functional $G : Rel \to Rel$ is defined, for each $\Re \in Rel$, as follows: $(s_1, s_2) \in G(\Re)$, if for each $a \in A$,

   i) $s_1 \xrightarrow{\langle a,n\rangle} s_1'$ implies $s_2 \xrightarrow{\langle a,m\rangle} s_2'$ such that $n \leq m$ and $(s_1', s_2') \in \Re$;

   ii) $s_2 \xrightarrow{\langle a,n\rangle} s_2'$ implies $s_1 \xrightarrow{\langle a,m\rangle} s_1'$ such that $n \geq m$ and $(s_1', s_2') \in \Re$.

2) A relation $\Re \in Rel$ will be called a $G - bisimulation$ if $\Re \subseteq G(\Re)$.

3) We say that two states s_1 and s_2 are in the relation of *f-performance preorder* (denoted $s_1 \sqsubseteq^f s_2$) if and only if there exists a $G - bisimulation$ $\Re$ such that $(s_1, s_2) \in \Re$.

4) We say that $E_1 \sqsubseteq_p^f E_2$ if and only if $0 \Rightarrow E_1 \sqsubseteq^f 0 \Rightarrow E_2$.

Whenever clear from the context we will omit the superscript duration function. Hence, we will often write $s_1 \sqsubseteq s_2$ or $E_1 \sqsubseteq_p E_2$.

Example 4.1 A few examples are now in order.

$$\begin{array}{rcl@{\qquad}rcl}
a \mid b & \sqsubseteq_p & a.b + b.a, & a.b + b.a & \not\sqsubseteq_p & a \mid b \\
a \mid b + a.b + b.a & \sqsubseteq_p & a.b + b.a, & a.b + b.a & \not\sqsubseteq_p & a \mid b + a.b + b.a \\
a \mid b & \sqsubseteq_p & a \mid b + a.b + b.a, & a \mid b + a.b + b.a & \not\sqsubseteq_p & a \mid b
\end{array}$$

Relation $\sqsubseteq_p$ is reflexive and transitive but not symmetric as the examples above show. Hence, $\sqsubseteq_p$ is a preorder indeed.

When relating processes with respect to speed by means of the performance preorder, of each set of *functionally equivalent computations* we focus only on the fastest one and on the slowest one, which provide a lower and upper bound to the time needed for executions. For instance, in $a \mid b + a.b + b.a$ there are two different computations for the sequence ab, with different temporal behaviour. We have that $E_1 = a|b+a.b+b.a \sqsubseteq_p a.b+b.a = E_2$ because the fastest computation for ab in E_1 is faster than the fastest (actually unique) ab computation from E_2, as well as the slowest computation for ab is faster than the slowest ab computation from E_2. All the "performance-intermediate" functionally-equivalent computations are irrelevant, as the following example shows.

Example 4.2 Consider the following two processes:

$$E_1 = a.a.a \; + \; a|a|a \qquad \text{and} \qquad E_2 = a.a.a \; + \; a|a|a \; + \; a.a|a$$

We have that $E_1 \sqsubseteq_p E_2$ and $E_2 \sqsubseteq_p E_1$. This shows that the "intermediate" aaa computations due to subagent $a.a|a$ have no influence on the preorder.

Similar properties of congruence of performance equivalence hold also for performance preorder.

Proposition 4.4 Given two performance preorder agents E_1 and E_2, $E_1 \sqsubseteq_p E_2$, we have that:

i) $a.E_1 \sqsubseteq_p a.E_2$ for any action $a \in Act$;

ii) $E + E_1 \sqsubseteq_p E + E_2$ and $E_1 + E \sqsubseteq_p E_2 + E$ for any agent E;

iii) $E_1 \mid E \sqsubseteq_p E_2 \mid E$ and $E \mid E_1 \sqsubseteq_p E \mid E_2$, for any agent E;

Proof: We only prove item iii); the proof for all other cases are similar if not simpler or follow standard techniques [18]. We know that $E_1 \sqsubseteq_p^f E_2$ iff $0 \Rightarrow E_1 \sqsubseteq^f 0 \Rightarrow E_2$. Clearly $0 \Rightarrow E \sqsubseteq^f 0 \Rightarrow E$. To prove that $0 \Rightarrow (E_1 \mid E) \sqsubseteq^f 0 \Rightarrow (E_2 \mid E)$ by the clock distribution equations, it suffices to show that $0 \Rightarrow E_1 \mid 0 \Rightarrow E \sqsubseteq^f 0 \Rightarrow E_2 \mid 0 \Rightarrow E$. This immediately follows by the statement:

$$s_1 \sqsubseteq^f s_2 \text{ and } s_3 \sqsubseteq^f s_4 \text{ implies } s_1 \mid s_3 \sqsubseteq^f s_2 \mid s_4$$

as the relation $\Re = \{\langle s_1 \mid s_3, s_2 \mid s_4 \rangle \mid s_1 \sqsubseteq^f s_2 \text{ and } s_3 \sqsubseteq^f s_4\}$ is a $G - bisimulation$. □

If we consider also open terms, it can be proved that $\sqsubseteq_p$ is a congruence also for recursion following standard techniques [18].

Relation $\sqsubseteq_p^f$ does depend on the choice of the action duration function f, as the following example shows.

Example 4.3 Consider the following pair of processes:

$$E_1 \;=\; a.a \mid c \quad \text{and} \quad E_2 \;=\; a.a \mid c \;+\; c.(a \mid a)$$

We can show that $E_1 \sqsubseteq_p^f E_2$ if and only if $f(a) \leq f(c)$. In fact every transition of $0 \Rightarrow E_1$ can be matched (in the same time) by a transition of $0 \Rightarrow E_2$. Similarly, transitions performed by the summand $0 \Rightarrow a.a \mid c$ of E_2 can be matched by transitions of $0 \Rightarrow E_1$. Consider now the transitions from $0 \Rightarrow c.(a \mid a)$. This term can perform $0 \Rightarrow c.(a \mid a) \xrightarrow{\langle c, f(c) \rangle} f(c) \Rightarrow a \mid f(c) \Rightarrow a$ and, in order to match this transition, $0 \Rightarrow E_1$ can only perform $0 \Rightarrow E_1 \xrightarrow{\langle c, f(c) \rangle} 0 \Rightarrow a.a \mid f(c) \Rightarrow nil$. Now, $0 \Rightarrow a.a \mid f(c) \Rightarrow nil \sqsubseteq^f f(c) \Rightarrow a \mid f(c) \Rightarrow a$ if and only if $f(a) \leq f(c)$ because the second a of the left state will be completed after $2 \times f(a)$ units of time, while the second a of the right one after $f(c) + f(a)$.

We want to remark that one could define a finer notion of performance preorder which is independent of the chosen duration function.

Definition 4.3 E_1 is related to E_2 by *independent-performance preorder*, denoted $E_1 \sqsubseteq_p^i E_2$, if for each f, $E_1 \sqsubseteq_p^f E_2$.

As an instance, consider Example 4.2: $E_1 \sqsubseteq_p^i E_2$ (and also the reverse holds). This is quite interesting, because we know that E_1 is always more efficient than E_2, whatever is the machine on which it will be implemented.

4.3 Competitive Equivalence

In a natural way, the kernel of the preorder gives rise to a notion of equivalence, called *competitive equivalence*.

Definition 4.4 Two agents E_1 and E_2 are *competitive* equivalent (denoted $E_1 \simeq_p E_2$) if $E_1 \sqsubseteq_p E_2$ and $E_2 \sqsubseteq_p E_1$. In other words, $\simeq_p = \sqsubseteq_p \cap \sqsupseteq_p$.

Analogously to performance preorder, competitive equivalence discriminates agents only by considering the fastest computation and the slowest one for any set of functionally-equivalent computations. Two agents are equivalent if these upper and lower bounds coincide. Also in this case, all the "performance-intermediate" functionally-equivalent computations are irrelevant, as Example 4.2 shows: $E_1 \simeq_p E_2$ even if E_2 has additional intermediate aaa computations.

Relation $\simeq_p$ enjoys the same properties of congruence enjoyed by the preorder. In particular it is preserved by every syntactic construct of the language, including recursion. Moreover, similarly to the preorder, competitive equivalence does depend on the duration assigned to every action.[4]

Proposition 4.5 Let E_1 and E_2 be two agents and f be a duration function. Then $E_1 \simeq_p^f E_2$ does not imply $E_1 \simeq_p^g E_2$ for all duration functions g.

Proof: Consider the following pair of processes:

$$E_1 \;=\; a.a \mid c \;+\; c.a.a \quad \text{ and } \quad E_2 \;=\; a.a \mid c \;+\; c.a.a \;+\; c.(a \mid a)$$

then we can show that $E_1 \simeq_p^f E_2$ if and only if $f(a) \leq f(c)$. To prove $E_1 \simeq_p^f E_2$ we have to show both $E_1 \sqsubseteq_p^f E_2$ and $E_2 \sqsubseteq_p^f E_1$. The latter immediately follows whatever is the time assigned to actions a and c. The former follows if $f(a) \leq f(c)$ by similar arguments of Example 4.3. $\square$

The same does not happen for the specific case of constant durational functions, as stated in the following proposition.

Proposition 4.6 Let E_1 and E_2 be two agents and f, g be any two constant duration functions. Then $E_1 \simeq_p^f E_2$ implies $E_1 \simeq_p^g E_2$.

Proof: Following similar reasonings of [3] (Proposition 13), where a similar proposition is proven for timing equivalence, it can be proved that performance preorder is preserved by constant duration functions. The result then immediately follows also for competitive equivalence. $\square$

We now compare performance equivalence and competitive equivalence. Because $=$ implies $\leq$ and $=$ implies $\geq$ in positive integers, $\sim_p$ is indeed a $G-bisimulation$. As a consequence of this observation we have the following:

Proposition 4.7 For each agent E_1 and E_2 and any duration function f, we have that $E_1 \sim_p^f E_2$ implies $E_1 \simeq_p^f E_2$.

The converse implication does not hold, i.e. $\simeq_p^f$ is strictly weaker than $\sim_p^f$. To this, consider the two agents in Example 4.2, which are $\simeq_p^f$-equivalent (for each duration function f) but not $\sim_p^f$-equivalent.

Definition 4.5 We say that E_1 is *independent-competitive equivalent* to E_2, denoted $E_1 \simeq_p^i E_2$, if $E_1 \sqsubseteq_p^f E_2$ and $E_2 \sqsubseteq_p^f E_1$ for each f.

It is worth noting that also $\simeq_p^i$ does not coincide with $\sim_p$. Again the pair of agents in Example 4.2 provides the required counter-example.

[4]This immediately distinguishes competitive equivalence from performance equivalence and timing equivalence [2, 3]. In the language considered in this paper, in fact, performance equivalence and timing equivalence do coincide; the latter does not depend on the chosen duration function, as proved in [3], Corollary 32 (in fact, on the finite sublanguage).

5 Proof Systems

The purpose of this section is to provide sound and complete axiomatizations for the performance preorder, as well as for competitive equivalence and performance equivalence, in the case of finite agents P_{fin}. We show that the basic set of axioms for performance equivalence can be extended with one inequational rule only, obtaining a sound and complete axiomatization for the preorder, hence also for competitive equivalence.

To this aim, we will take advantage of the operational semantics proposed in the previous section on the set of extended states, which include also finite agents. Therefore, we shall actually provide a sound and complete axiomatization of $\sqsubseteq_p$ over the set T_{fin} of finite extended states.

We follow Hennessy' and Milner's technique [17] which involves the reduction of states to normal forms, namely labelled trees. However, in our case, the labels are not simply actions, as for the interleaving approach, rather "decorated actions" of the form $\langle a, n\rangle$. To this we need to extend the algebra generating the language T_{fin} with two additional auxiliary operators, obtaining the language T'. These operators are a general form of prefixing with decorated actions $\langle a, n\rangle : t$ and a left-merge operator $t_1 \rfloor t_2$. The operational semantics for T' is obtained by extending the rules in Table 3 with those reported in Table 5.

Pre	$\dfrac{}{\langle a,n\rangle : t \xrightarrow{\langle a,n\rangle} t}$	$Lpar$	$\dfrac{t_1 \xrightarrow{\langle a,n\rangle} t_1'}{t_1 \rfloor t_2 \xrightarrow{\langle a,n\rangle} t_1' \vert t_2}$

Table 5: Operational Rules for Auxiliary Operators.

These auxiliary operators are trivially preserved by strong bisimulation; hence, performance equivalence is a congruence also for them. The generalized prefix is necessary in order to define a suitable notion of normal form for terms in T', while the left-merge is needed to define a finite set of axioms reducing any state t to a normal form.

Definition 5.1 A *normal form* is a term of the form:

$$\sum_{i \in I} \langle a_i, n_i\rangle : t_i$$

where $\forall i \in I$, t_i is a normal form, $a_i \in A$ and n_i is a positive integer. By convention this normal form is *nil* if $I = \emptyset$.

Let $\mathcal{A}$ be the set of axioms reported in Table 6, which forms a sound and complete axiomatization for performance equivalence. This equivalence already received an axiomatisation on the same language we consider [3, 25]. However, our present formulation has the merit of being *scalable* to the case of performance preorder and competitive equivalence, we will give immediately after.

With the expression $\mathcal{A} \vdash t_1 = t_2$ we mean that the two terms t_1 and t_2 are proved equal using the axioms in the set $\mathcal{A}$ and the usual inference rules of reflexivity, symmetry, transitivity and substitutivity of $=$.

Substitutivity is the characterizing property of congruence relations. It states that the operation of inserting equal terms in the same context gives rise to equal terms. One particular instance of this fact is reported below

$$(Sub) \quad t_1 = t_2 \text{ implies } \langle a,n\rangle : t_1 \;=\; \langle a,n\rangle : t_2 \quad \text{with } a \in Act \text{ and } n \text{ a positive integer.}$$

The following proposition states the soundness of axioms in Table 6 w.r.t. performance equivalence, hence also for competitive equivalence and performance preorder.

$(sum1)$	$t_1 + t_2$	$=$	$t_2 + t_1$
$(sum2)$	$(t_1 + t_2) + t_3$	$=$	$t_1 + (t_2 + t_3)$
$(sum3)$	$t + t$	$=$	t
$(sum4)$	$t + nil$	$=$	t
$(pref)$	$a.E$	$=$	$\langle a, f(a)\rangle : (f(a) \Rightarrow E)$
(exp)	$t_1 \vert t_2$	$=$	$t_1 \rfloor t_2 + t_2 \rfloor t_1$
$(lpar1)$	$nil \rfloor t$	$=$	nil
$(lpar2)$	$(\langle a, k\rangle : t_1) \rfloor t_2$	$=$	$\langle a, k\rangle : (t_1 \vert t_2)$
$(lpar3)$	$(t_1 + t_2) \rfloor t_3$	$=$	$(t_1 \rfloor t_3) + (t_2 \rfloor t_3)$
$(clock1)$	$n \Rightarrow nil$	$=$	nil
$(clock2)$	$n \Rightarrow \langle a, k\rangle : t$	$=$	$\langle a, k+n\rangle : (n \Rightarrow t)$
$(clock3)$	$n \Rightarrow (t_1 + t_2)$	$=$	$(n \Rightarrow t_1) + (n \Rightarrow t_2)$

Table 6: The Axiom Set $\mathcal{A}$

Proposition 5.1 For each t_1, $t_2 \in T'$, $\mathcal{A} \vdash t_1 = t_2$ implies $t_1 \sim_p t_2$.

Proof: The proof is standard. We just show the soundness for two interesting axioms: $(pref)$ and $(clock2)$.

- Axiom $(pref)$ is sound. To prove this it suffices to observe that relation

$$\Re = \{ \, (\, a.E, \langle a, f(a)\rangle : (f(a) \Rightarrow E) \,) \, \} \; \cup id$$

is an $R-bisimulation$. In fact id (denoting the identity relation) is clearly an $R-bisimulation$ and $a.E \stackrel{\langle a,f(a)\rangle}{\longrightarrow} (f(a) \Rightarrow E)$ iff $\langle a, f(a)\rangle : (f(a) \Rightarrow E) \stackrel{\langle a,f(a)\rangle}{\longrightarrow} (f(a) \Rightarrow E)$.

- Axiom $(clock2)$ is sound. In this case, relation

$$\Re = \{ \, (\, n \Rightarrow \langle a, k\rangle : t, \; \langle a, k+n\rangle : (n \Rightarrow t) \,) \, \} \; \cup id$$

is an $R - bisimulation$. To this note that id is an $R - bisimulation$ and $n \Rightarrow \langle a, k\rangle : t \stackrel{\langle a,n+k\rangle}{\longrightarrow} n \Rightarrow t$ iff $\langle a, n+k\rangle : (n \Rightarrow t) \stackrel{\langle a,n+k\rangle}{\longrightarrow} n \Rightarrow t$.

□

To prove completeness of the axioms in $\mathcal{A}$ w.r.t. performance equivalence, we first show that each state can be reduced to a normal form. We need the auxiliary definition of size of a term, together with two obvious properties. The *size* of an extended term is defined by structural induction by:

$$\begin{aligned}
size(nil) &= 0 \\
size(a.E) &= 1 + size(E) \\
size(\langle a, n\rangle : t) &= 1 + size(t) \\
size(t_1 + t_2) &= max\{size(t_1), size(t_2)\} \\
size(t_1 \vert t_2) &= size(t_1) + size(t_2) \\
size(t_1 \rfloor t_2) &= size(t_1) + size(t_2) \\
size(n \Rightarrow t) &= size(t).
\end{aligned}$$

Proposition 5.2 For each $t \in T'$, we have that $size(t) \geq 0$.

Proposition 5.3 For each equation $t = t' \in \mathcal{A}$ (with the order as represented in Table 6), we have that $size(t) \geq size(t')$.

The two propositions above are used (implicitly) in several places during the proof of the "normalization" proposition below. In particular, axiom $(lpar1)$ is used always to rewrite a term in such a way that the size decreases.

Proposition 5.4 For each $t \in T'$ there exists a normal form $nf(t)$ such that $\mathcal{A} \vdash t = nf(t)$.

Proof: The proof proceeds by induction on the size of t. Assume the proposition true for terms t with $size(t) < n$. We prove it for terms t with $size(t) = n$. The proof proceeds by (inner) induction on the syntactic structure of terms.

(a) $t = nil$. Clearly $nf(t) = nil$.

(b) $t = a.E$. By axiom $(pref)$, $a.E = \langle a, f(a)\rangle : (f(a) \Rightarrow E)$. Since $size(f(a) \Rightarrow E) < size(a.E)$, the inductive hypothesis ensures that $f(a) \Rightarrow E$ can be reduced to a normal form $nf(f(a) \Rightarrow E)$ such that $\mathcal{A} \vdash (f(a) \Rightarrow E) = nf(f(a) \Rightarrow E)$. By substitutivity it follows that $\mathcal{A} \vdash \langle a, f(a)\rangle : (f(a) \Rightarrow E) = \langle a, f(a)\rangle : nf(f(a) \Rightarrow E)$ which is a normal form for $a.E$.

(c) $t = t_1 + t_2$. By structural induction, there are two normal forms $nf(t_1)$, $nf(t_2)$ such that $\mathcal{A} \vdash t_1 = nf(t_1)$ and $\mathcal{A} \vdash t_2 = nf(t_2)$. If $nf(t_1) = nil$ then by axioms $(sum1)$ and $(sum4)$, $nf(t_2)$ is the normal form for t. Symmetrically, if $nf(t_2) = nil$, then $nf(t_1)$ is the normal form for t by axiom $(sum4)$. Otherwise if $nf(t_1) \neq nil$ and $nf(t_2) \neq nil$ then $nf(t_1) + nf(t_2)$ is the normal form for t (up to $(sum1)$, $(sum2)$ and $(sum3)$).

(d) $t = t_1 \rfloor t_2$. By structural induction, there is normal form $nf(t_1)$ such that $\mathcal{A} \vdash t_1 = nf(t_1)$. If $nf(t_1) = nil$ then by axiom $(lpar1)$, nil is the normal form for t. Otherwise, if $nf(t_1) \neq nil$, by substitutivity $\mathcal{A} \vdash t_1 \rfloor t_2 = nf(t_1) \rfloor t_2 = (\sum_{i \in I} \langle a_i, n_i\rangle : t_i) \rfloor t_2 =$ by axiom $(lpar3) = \sum_{i \in I}((\langle a_i, n_i\rangle : t_i) \rfloor t_2) =$ by axiom $(lpar2) = \sum_{i \in I} \langle a_i, n_i\rangle : (t_i \mid t_2)$. Since $size(t_i \mid t_2) < size(t_1 \rfloor t_2)$, it follows, by inductive hypothesis on the size of terms, that there are normal forms $nf(t_i \mid t_2)$ such that $\mathcal{A} \vdash t_i \mid t_2 = nf(t_i \mid t_2)$. Thus, by substitutivity, $\mathcal{A} \vdash t_1 \rfloor t_2 = \sum_{i \in I} \langle a_i, n_i\rangle : nf(t_i \mid t_2)$, which is a normal form.

(e) $t = t_1 \mid t_2$. By structural induction (note that $size(t_1) \leq n$ and $size(t_2) \leq n$), there are two normal forms $nf(t_1)$, $nf(t_2)$ such that $\mathcal{A} \vdash t_i = nf(t_i)$ for $i = 1, 2$. By substitutivity, $\mathcal{A} \vdash t_1 \mid t_2 = nf(t_1) \mid nf(t_2)$. By axiom (exp), $\mathcal{A} \vdash nf(t_1) \mid nf(t_2) = nf(t_1) \rfloor nf(t_2) + nf(t_2) \rfloor nf(t_1)$ and by (d) there are two normal forms nf_1 and nf_2 such that $\mathcal{A} \vdash nf(t_1) \rfloor nf(t_2) = nf_1$ and $\mathcal{A} \vdash nf(t_2) \rfloor nf(t_1) = nf_2$. Thus, by (c), $nf(t)$ is either nf_1 (if $nf(t_2) = nil$), or nf_2 (if $nf(t_1) = nil$) or $nf_1 + nf_2$.

(f) $t = n \Rightarrow t_1$. By structural induction hypothesis, there exists a normal form $nf(t_1)$ such that $\mathcal{A} \vdash t_1 = nf(t_1)$. If $nf(t_1) = nil$ then by axiom $(clock1)$, nil is the normal form for t. Otherwise, $\mathcal{A} \vdash (n \Rightarrow t_1) = (n \Rightarrow nf(t_1)) = (n \Rightarrow (\sum_{i \in I} \langle a_i, n_i\rangle : t_i)) =$ by axiom $(clock3)$ $= \sum_{i \in I}(n \Rightarrow (\langle a_i, n_i\rangle : t_i)) =$ by axiom $(clock2)$ $= \sum_{i \in I} \langle a_i, n_i + n\rangle : (n \Rightarrow t_i)$. By induction on the size, there are normal forms $nf(n \Rightarrow t_i)$ such that $\mathcal{A} \vdash (n \Rightarrow t_i) = nf(n \Rightarrow t_i)$. By substitutivity $\mathcal{A} \vdash (n \Rightarrow t_1) = \sum_{i \in I} \langle a_i, n_i + n\rangle : nf(n \Rightarrow t_i)$ which is a normal form.

$\square$

The completeness theorem follows.

Theorem 5.1 For each t_1, $t_2 \in T'$, $t_1 \sim_p t_2$ implies $\mathcal{A} \vdash t_1 = t_2$.

Proof: By Proposition 5.4 we can assume that both t_1, t_2 are in normal form, namely $nf(t_1)$ and $nf(t_2)$ respectively. The proof proceeds by induction on the sum of their sizes. Clearly if the size of $nf(t_1)$ and $nf(t_2)$ are 0 the result is trivial. Otherwise assume that $nf(t_1) = \sum_{i\in I}\langle a_i, n_i\rangle : nf'_i$ and $nf(t_2) = \sum_{j\in J}\langle b_j, m_j\rangle : nf''_j$. We first prove that $nf(t_1) + nf(t_2) = nf(t_2)$ and then that $nf(t_1) + nf(t_2) = nf(t_1)$. Thus, by transitivity it follows that $nf(t_1) = nf(t_2)$.

As $nf(t_1) \sim_p nf(t_2)$, we have that for each summand $\langle a_i, n_i\rangle : nf'_i$ of $nf(t_1)$ there exists $h(i) \in J$ such that $\langle b_{h(i)}, m_{h(i)}\rangle : nf''_{h(i)}$ is a summand of $nf(t_2)$ such that $a_i = b_{h(i)}$, $n_i = m_{h(i)}$ and $nf'_i \sim_p nf''_{h(i)}$. By induction hypothesis $\mathcal{A} \vdash nf'_i = nf''_{h(i)}$, and thus by rule (Sub) we have that $\mathcal{A} \vdash \langle a_i, n_i\rangle : nf'_i = \langle b_{h(i)}, m_{h(i)}\rangle : nf''_{h(i)}$. This is the case for each summand of $nf(t_1)$, thus we obtain by substitutivity:

$$\mathcal{A} \vdash nf(t_1) = \sum_{i\in I}\langle a_i, n_i\rangle : nf'_i = \sum_{i\in I}\langle b_{h(i)}, m_{h(i)}\rangle : nf''_{h(i)}$$

Hence $nf(t_1) + nf(t_2) = \sum_{i\in I}\langle b_{h(i)}, m_{h(i)}\rangle : nf''_{h(i)} + nf(t_2) = nf(t_2)$ by substitutivity and axiom $(sum3)$.

Symmetrically, we can prove that $nf(t_1) + nf(t_2) = nf(t_1)$, from which the thesis follows. □

To give a sound and complete axiomatization of $\sqsubseteq_p$ over T', we need the introduction of inequations. In particular, we have inequations of the form $t_1 \leq t_2$ whenever $t_1 = t_2$ is an equation in $\mathcal{A}$, producing a set $\mathcal{A}_\leq$ of inequations. The usual inference rules of reflexivity, transitivity and substitutivity of $\leq$ are also coupled with the usual rules for equality, saying that:

$$\frac{t_1 \leq t_2 \quad \text{and} \quad t_2 \leq t_1}{t_1 = t_2} \qquad \frac{t_1 = t_2}{t_1 \leq t_2 \quad \text{and} \quad t_2 \leq t_1} \qquad \frac{t_1 = t_2}{t_2 = t_1}$$

Note, however, that $\mathcal{A} \vdash t_1 = t_2$ if and only if $\mathcal{A}_\leq \vdash t_1 \leq t_2$ and $\mathcal{A}_\leq \vdash t_2 \leq t_1$. The substitutivity of $\leq$ w.r.t. the extended prefix reads as

$$(Sub') \quad t_1 \leq t_2 \text{ implies } \langle a, n\rangle : t_1 \leq \langle a, n\rangle : t_2.$$

The set $\mathcal{A}_\leq$ is enriched with one single inequational rule:

$$\boxed{(Sub_\leq) \quad n \leq m \text{ and } t_1 \leq t_2 \quad \text{imply} \quad \langle a, n\rangle : t_1 \leq \langle a, m\rangle : t_2}$$

This essentially extends (Sub') by permitting the clock value of the former to be less than or equal to the one of the latter (condition $n \leq m$).

Let $\mathcal{B}_\leq$ be the set $\mathcal{A}_\leq \cup \{Sub_\leq\}$. We want to prove that $\mathcal{B}_\leq$ is a sound and complete algebraic characterization of the performance preorder and competitive equivalence.

Proposition 5.5 For each t_1, $t_2 \in T'$, we have the following:

i) $\mathcal{B}_\leq \vdash t_1 \leq t_2$ implies $t_1 \sqsubseteq_p t_2$, and

ii) $\mathcal{B}_\leq \vdash t_1 = t_2$ implies $t_1 \simeq_p t_2$.

Proof: Standardly we can prove soundness of axioms by exhibiting appropriate $G-bisimulations$. Here we just show the soundness of the new rule $(Sub_\leq)$. To this end, we have to prove that if $n \leq m$ and $t_1 \leq t_2$, then $\langle a, n\rangle : t_1 \leq \langle a, m\rangle : t_2$. This simply follows by observing that if $\Re'$ is a $G-bisimulation$ containing (t_1, t_2), then relation $\Re = \{\, (\langle a, n\rangle : t_1, \langle a, m\rangle : t_2) \,\} \cup \Re'$ is also $G-bisimulation$ for $(\langle a, n\rangle : t_1, \langle a, m\rangle : t_2)$. □

To prove completeness of the equations and inequation in $\mathcal{B}_{\leq}$ w.r.t. performance preorder and competitive equivalence, we first recall that each state can be reduced to a normal form using the equations in $\mathcal{A}$ only (Proposition 5.4).

Theorem 5.2 For each t_1, $t_2 \in T'$, we have the following:

- $t_1 \sqsubseteq_p t_2$ implies $\mathcal{B}_{\leq} \vdash t_1 \leq t_2$, and
- $t_1 \simeq_p t_2$ implies $\mathcal{B}_{\leq} \vdash t_1 = t_2$.

Proof: The completeness of the *performance preorder* follows by similar lines of Theorem 5.1, where every $=$ is replaced by $\leq$, every $\sim_p$ is replaced by $\sqsubseteq_p$ and every (Sub) is replaced by $(Sub_{\leq})$. Thus we assume that both t_1 and t_2 are in normal form, namely $nf(t_1)$ and $nf(t_2)$, and then we show that $nf(t_1) \ \leq \ nf(t_1) \ + \ nf(t_2)$ and $nf(t_1) \ + \ nf(t_2) \ \leq \ nf(t_2)$. By transitivity it follows that $nf(t_1) \ \leq \ nf(t_2)$.

The completeness of competitive equivalence immediately follows by the completeness of the performance preorder and rule "$t_1 \ \leq \ t_2$ and $t_2 \ \leq \ t_1$ imply $t_1 \ = \ t_2$". □

6 Comparative Performance Semantics

In this section we first show how our preorder relates with *strong bisimulation equivalence*. Then we compare our preorder with *pomset preorder* [1] and *location preorder* [6]. Finally, we establish how competitive equivalence relates with the best known untimed non-interleaving semantics proposed in the literature, namely *location* [6], *causal* [10], *ST* [15, 13] and *step* [23] equivalences.

6.1 Relationships with Strong Bisimulation Equivalence

Performance preorder enjoys some interesting properties w.r.t. bisimulation equivalence $\sim$, summarized in the following theorem (where P_{seq} is the set of processes built without parallel composition).

Theorem 6.1 Let E_1, $E_2 \in P, E_3 \in P_{seq}$ and f is an action duration function. Then:

1) $E_1 \ \sqsubseteq_p^f \ E_2$ implies $E_1 \sim E_2$

2) $E_1 \ \sim \ E_3$ iff $E_1 \sqsubseteq_p^f E_3$

Proof: We just sketch the proof. We need to introduce two new functions over states: *cut* and *clocks*. Taken a state s, function *cut* returns the agent E obtained by forgetting all time prefixings $n \Rightarrow \_$ from s. Taken a state s, function *clocks* returns the set $\{n_1, ..., n_k\}$ of all clock values n_i, occurring in time prefixing $n_i \Rightarrow \_$ within s. These functions can be defined by induction on the syntactic structure of states and it is possible to prove the following statement: For all s, $s' \in S$: $s \xrightarrow{\langle a,n \rangle} s'$ iff $cut(s) \xrightarrow{a} cut(s')$, where $\xrightarrow{a}$ denotes the standard interleaving operational semantics [18].

To prove (1) and (2), we show that:

(1) $E_1 \ \sqsubseteq_p^f \ E_2$ implies $E_1 \sim E_2$ because relation

$$\Re \ = \ \{(cut(s_1'), cut(s_2')) \mid s_1' \ \sqsubseteq_p^f \ s_2'\}$$

is a strong bisimulation containing (E_1, E_2).

(2) $(\Rightarrow)$ $E_1 \ \sim \ E_3$ implies $E_1 \sqsubseteq_p^f E_3$ because relation

$$\Re \ = \ \{(s, n \Rightarrow E_3') \mid \ cut(s) \ = \ E_2', \ E_1' \ \sim \ E_2' \text{ and } n \geq max(clocks(s))\}$$

is a $G - bisimulation$ containing $(0 \Rightarrow E_1, 0 \Rightarrow E_3)$.

($\Leftarrow$) It follows by (1).

$\square$

Item 2) can be sometimes useful because, as noted in [1], refinements of specifications into implementations frequently proceed by substituting parallel processes for sequential nondeterministic ones in a context. Hence in most practical cases checking that $Imp \sqsubseteq_p^f Spec$ reduces to showing bisimilarity between sequential nondeterministic subparts of $Spec$ and their parallel implementations.

6.2 Relationships with Non-interleaving Preorders

We will consider two non-interleaving preorders. Firstly, the pomset preorder of [1] and then the location preorder of [6]. The former permits to relate concurrency and nondeterminism without reducing the former to the latter. We show that there is no immediate relationship between the pomset preorder and ours. To this, we first prove that our preorder can relate processes with different causal dependencies between their actions. For the vice versa we recall an example of processes showing that pomset preorder does not respect the "branching-time structure". The location preorder, instead, is based on a simple intuition: two processes are related if one is "more distributed" over the space than the other one. We show that the location preorder strictly implies the performance preorder so that if E_1 is "more distributed than" E_2 then E_1 is "faster than" E_2, but the vice versa is not always true. Surprisingly enough, we provide an example showing that the location preorder fails to capture some intuitive "increase of distribution" that the performance preoder, instead, is able to represent.

6.2.1 Pomset Preorder

In this section we just sketch the intuition behind pomset preorder and refer the reader to [1] for more details. In the semantic theory of [1], processes evolve by performing transitions of the form $p_1 \xrightarrow{u} p_2$ where p_1 and p_2 are P terms while u is a pomset of actions in Act, called *computation*. The tool for relating non determinism and concurrency in the behavior of the processes in P will be a preorder over computations $\leq_C$. Intuitively, for computations u and v, $u \leq_C v$ is intended to capture the fact that u and v correspond to the execution of the same actions, but these actions are performed in a less caused fashion in the computation v. Formally, let $\leq_C$ denote the least precongruence over the set of computations $Comp$ which satisfies axioms C in Table 7.

(PAR_1)	$x \mid nil$	$=$	x
(PAR_2)	$x \mid y$	$=$	$y \mid x$
(PAR_3)	$(x \mid y) \mid z$	$=$	$x \mid (y \mid z)$
(SEQ)	$a.(x \mid y)$	$\leq$	$a.x \mid y$

Table 7: The Preorder on Computations.

The syntax preorder $\leq_C$ over $Comp$ is used to induce a preorder, called pomset preorder and denoted $\sqsubseteq_c$, over the set of processes P. This preorder does not reduce concurrency to sequentiality and nondeterminism. Its definition is similar to that of performance preorder but instead of relating natural numbers, pomset preorder relates pomsets with respect to $\leq_C$. The following proposition shows that $\sqsubseteq_c$ is incomparable with our performance preorder.

Proposition 6.1 Let E_1 and E_2 be two P terms. Then:

$$\begin{array}{lllll} \text{i)} & E_1 \sqsubseteq_c E_2 & \text{does not imply} & E_1 \sqsubseteq_p^f E_2 & \text{or} \quad E_1 \sqsupseteq_p^f E_2, \\ \text{ii)} & E_1 \sqsubseteq_p^f E_2 & \text{does not imply} & E_1 \sqsubseteq_c E_2 & \text{or} \quad E_1 \sqsupseteq_c E_2. \end{array}$$

Proof: We provide a pair of processes related by $\sqsubseteq_c$ but not by $\sqsubseteq_p^f$ or $\sqsupseteq_p^f$ and vice versa.

i) $a.b + b.a \sqsubseteq_c a \mid b$ but $a.b + b.a \not\sqsubseteq_p^f a \mid b$. Moreover,

$a.(b+c) + a \mid b \sqsubseteq_c a.(b+c) + a \mid b + a.b$ but $a.(b+c) + a \mid b \not\sqsupseteq_p^f a.(b+c) + a \mid b + a.b$ whatever is the duration function f.

ii) $a \mid b \sqsubseteq_p^f a.b + b.a$ for each f but $a \mid b \not\sqsubseteq_c a.b + b.a$. Moreover, $a.a|a.a \sqsubseteq_p^f a.a.(a|a)$ for any f, but $a.a.(a|a) \not\sqsubseteq_c a.a|a.a$. To show that $a.a.(a|a) \not\sqsubseteq_c a.a|a.a$ consider the transitions $a.a|a.a \xrightarrow{a.a} nil|a.a \xrightarrow{a.a} nil|nil$ which cannot be successfully matched by $a.a.(a|a)$.

□

For similar reasons the kernel of the pomset preorder, $\simeq_c$, does not coincide with our competitive equivalence. Processes $E_1 = a.(b+c) + a|b + a.b$ and $E_2 = a.(b+c) + a|b$ are $\simeq_c$-equivalent but not competitive equivalent (note that $E_1 \not\sqsubseteq_p^f E_2$). This example shows also that $\simeq_c$ does not completely respect the branching-time structure of processes. The first agent may "cause" via an a either $(b+c)$ or just b, while the second one has no choice.

To prove that $\simeq_p^f$ does not imply $\simeq_c$, consider the pair of processes $E_1 = a.a.a.a.a.a + a.a|a.a|a.a$ and $E_2 = a.a.a.a.a.a + a.a|a.a|a.a + a.a|a.a.(a|a)$ which are $\simeq_p^f$-equivalent (for each f) but not $\simeq_c$-equivalent (note that $E_2 \not\sqsubseteq_c E_1$, because $a.a|a.a.(a|a) \not\sqsubseteq_c a.a|a.a|a.a$). It follows that $\simeq_p^f$ and $\simeq_c$ are incomparable.

Proposition 6.2 Pomset preorder and performance preorder coincide over P_{seq}.

Proof: It follows by Theorem 6.1.(2) and because strong observational equivalence over P_{seq} coincides with pomset preorder, as proven in [1]. □

6.2.2 Location Preorder

In this section we just sketch the intuition behind location preorder and refer the reader to the Appendix for more details.

To detect the distributed nature of processes, location names taken from an infinite set Loc of names, are dynamically assigned to performed actions. This is achieved by means of a transition relation of the form $\xrightarrow{a,u}$ where a is an action, $u = l_1 ... l_n l$ is a word over the set of locations representing the access path to the parallel component executing the a and l is a new location generated by the observation of a. The rules central to this notion of location are:

$$\frac{}{a.p \xrightarrow{a,l} l :: p} \quad \text{and} \quad \frac{p \xrightarrow{a,u} p'}{l :: p \xrightarrow{a,lu} l :: p'} ,$$

for any location l, where $l :: \_$ is a new location prefixing operator. Location preorder is based on the notion of subword. For $u, v \in Loc^*$, $u \leq v$, denotes that u is a subword of v, i.e. $u = l_1 l_2 ... l_k$, where $l_i \in Loc$ and there exists $v_1 ... v_{k+1} \in Loc^*$ such that $v_1 l_1 v_2 l_2 ... v_{k-1} l_k v_k = v$.

This preorder $\leq$ on location names induces a preorder $\sqsubseteq_l$ on processes. Its definition is very similar to that of performance preorder but instead of relating natural numbers, it relates locations with respect to $\leq$. We show that location preorder strictly implies performance preorder so that if E_1 is "more distributed than" E_2, then E_1 is "faster than" E_2, but the vice versa is not always true.

Proposition 6.3 Let E_1 and E_2 be P processes.

i) $E_1 \sqsubseteq_l E_2$ implies $E_1 \sqsubseteq_p^i E_2$;

ii) $E_1 \sqsubseteq_p^i E_2$ does not imply $E_1 \sqsubseteq_l E_2$.

Proof: The proof of i) is reported in the Appendix. To see ii) let us consider the pair of processes $E_1 = a.a|a.a$ and $E_2 = a.a.(a|a)$ given in Proposition 6.1 ii) which are $\sqsubseteq_p^i$-related but not $\sqsubseteq_l$-related. To show that $E_1 \not\sqsubseteq_l E_2$ note that the sequence of transitions $E_1 \xrightarrow{a,l_1} l_1 :: a|a.a \xrightarrow{a,l_1l_2} l_1 :: l_2 :: nil|a.a \xrightarrow{a,l_3} l_1 :: l_2 :: nil|l_3 :: a \xrightarrow{a,l_3l_4} l_1 :: l_2 :: nil|l_3 :: l_4 :: nil$ cannot be matched by corresponding transitions of E_2. □

The example reported in the proof above is very interesting as, intuitively, one should consider $a.a|a.a$ more distributed than $a.a.(a|a)$. The fact that the location preorder fails to relate these two terms is, in our opinion, a serious drawback of that preorder. Apparently, the performance preorder is, in this respect, more satisfactory.

The proposition below studies, instead, the relationships between the kernel of location preorder (defined as usual by $\simeq_l = \sqsubseteq_l \cap \sqsupseteq_l$) and competitive equivalence.

Proposition 6.4 Let E_1 and E_2 be P processes.

i) $E_1 \simeq_l E_2$ implies $E_1 \simeq_p^i E_2$;

ii) $E_1 \simeq_p^i E_2$ does not imply $E_1 \simeq_l E_2$.

Proof: i) It follows immediately by Proposition 6.3. To see ii) let us consider the pair of processes $E_1 = a.a.a.a + a.a|a.a$ and $E_2 = a.a.a.a + a.a|a.a + a.a.(a|a)$ which are $\simeq_p^i$-related but not $\simeq_l$-related (note that $E_1 \not\sqsubseteq_l E_2$, because $a.a|a.a \not\sqsubseteq_l a.a.(a|a)$. □

6.3 Relationships with Non-interleaving Equivalences

By establishing the relationships of competitive equivalence with performance equivalence (Proposition 4.7) we also understand how it relates with location equivalence (where actions – together with their localisation – are observed) [6], causal equivalence (where actions – together with their causes – are observed) [10] and ST-equivalence (where actions are not atomic and transitions are labeled by their beginnings or endings) [15].

Proposition 6.5 For processes in P, location equivalence (as well as causal equivalence and ST equivalence) strictly implies competitive equivalence.

Proof: Performance equivalence and timing equivalence [2, 3] trivially coincide over P. The latter in turn has been proved to coincide with location, causal and ST equivalences in [3] over P.[5] The result then follows by Proposition 4.7. □

It is worth noting that competitive equivalence is incomparable with *step equivalence* [23], the coarsest non-interleaving semantics proposed so far. Intuitively the transitions of the step

[5]To be precise, the proofs reported there are for the finite sublanguage. However, it is not difficult to see that the same hold also in presence of recursion (even if with a different proof technique).

semantics are labelled by multisets of actions which are concurrently executable. Thus for example $a|b \sim_{Step} a|b + a.b$. Clearly $a|b \not\simeq^f_p a|b + a.b$ for any f because $a|b + a.b \not\sqsubseteq^f_p a|b$. To see that $\simeq^f_p$ does not imply $\sim_{Step}$ consider the pair of processes E_1 and E_2 in Proposition 6.4 which are $\simeq^i_p$-equivalent but not $E_1 \sim_{Step} E_2$. In fact process $E_2 \xrightarrow{a} a.(a|a)$ while E_1 can perform a but the resulting a actions can be performed either sequentially only, or in a more parallel fashion.

Proposition 6.6 Step equivalence and competitive equivalence are incomparable.

7 A simple example

In this section we show how our theory can be applied to the stepwise refinement of a specification to its "optimal" implementation. Consider a simple distributed system server which is able to manage at most three requests from the external environment. It receives messages via a channel *in* and send acknowledgments of "received message" via a channel *ack*.

A possible specification of this simple server can be described in our language by:

$$Spec ::= \mathrm{rec}\ x.\ in.(\mathrm{rec}\ y.\ ack.x\ +\ in.(\mathrm{rec}\ z.\ ack.y\ +\ in.ack.z)).$$

Spec, completely sequential, represents also the behaviour of our server when executed over a uniprocessor architecture (where actions can be performed in interleaving way only). There are also two, more parallel, possibilities:

1) Two processors are available for its execution. Such a possible situation is that described by the following term:

$$Spec' ::= (\mathrm{rec}\ x.\ in.(\mathrm{rec}\ y.\ ack.x\ +\ in.ack.y))\ |\ \mathrm{rec}\ x.\ in.ack.x$$

2) The server can be mapped onto three different processors so that all its parallelism can be fired.

$$Imp ::= \mathrm{rec}\ x.\ in.ack.x\ |\ \mathrm{rec}\ x.\ in.ack.x\ |\ \mathrm{rec}\ x.\ in.ack.x.$$

It is really easy now to find $G - bisimulations$ to prove that (for any duration function f)

$$Imp\ \sqsubseteq^f_p\ Spec'\ \sqsubseteq^f_p\ Spec$$

that is process *Spec*, completely sequential, is slower than *Spec'* which explicates more parallelism. The latter, in turn, is slower than *Imp* which fires all possible parallelism of its actions. On the other hand it is possible to show that

$$Spec\ \not\sqsubseteq^f_p\ Spec' \quad \text{and} \quad Spec'\ \not\sqsubseteq^f_p\ Imp$$

that is the three processes are not competitive equivalent. It follows that, as expected, the performance of a process depends on the architecture on which it will be executed.

This simple example also motivates another concern: we could see *Spec* as the specification of our system server, *Imp* as the implementation and *Spec'* as a process obtained by expliciting parallelism in the specification. Thus our preorder supports the stepwise refinement "from specification to implementation" by retaining functional behaviour but increasing efficiency. This is not possible in the standard theory of bisimulation because of the semantic reduction of parallelism to sequentiality and nondeterminism and it is not possible with performance equivalence ($\sim^f_p$) because actions must be performed at the same time. Concerning the example above we would obtain:

$$Imp\ \sim\ Spec'\ \sim\ Spec \qquad \text{and} \qquad Imp\ \not\sim^f_p\ Spec'\ \not\sim^f_p\ Spec.$$

It can be proved that, if there exists a process Imp' such that $Imp' \sqsubseteq^f_p Imp$, then necessarily we also have that $Imp \sqsubseteq^f_p Imp'$. Hence, *Imp* is minimal in the performance ordering, i.e., it is an optimal implementation of *Spec*.

8 Conclusions

In this paper we have presented a semantic theory for a simple process algebra which is based on a preorder $\sqsubseteq_p$ relating two processes E_1 and E_2 if they have the same functional behaviour and E_1 is at least as fast as E_2. Briefly, we have shown that:

a) $\sqsubseteq_p$ is defined on the basis of well established and well known assumptions, as several papers have been published on process theory with durational actions and the present one naturally extends some of them.

b) $\sqsubseteq_p$ enjoys "good" mathematical properties; for instance, it is a congruence over the language we study.

c) $\sqsubseteq_p$ satisfies quite intuitive structural properties of processes; we show, at least for concurrent but not communicating processes, that the more distributed a process is the more efficient it is.

d) $\sqsubseteq_p$ permits to order processes by increasing efficiency while retaining the same functionality. And finally the paper presents:

e) A thorough comparison with other non-interleaving preorders and with related literature. This includes a formal comparison with Boudol, Castellani, Hennessy and Kiehn' location preorder, that, indeed, focuses on the distribution of processes. We show that in some cases our preorder captures more correctly the intuitive idea of distribution (see discussion after Proposition 6.3).

The following of this section will be devoted to some remarks on comparison with related work and on extension of our approach to a setting with communication, restriction and the invisible action. Finally, open problems and future research are hinted.

8.1 Related Work

Some other preorders relating processes with respect to speed have been proposed in the literature. All these approaches have in common with us only the goal because they conceive "time passing" in a completely different way.

In [4] a notion of efficiency is based on the number of communications that have taken place. Hence, E_1 is "faster" than E_2 if E_1 never needs to perform more internal (τ-) actions than E_2. As in our approach, the comparison is made by considering both the worst and optimal cases, while in [5] the same idea is followed but only the worst case is considered. In a similar line, a recent paper [22] studies testing based efficiency preorders. These notions of efficiency have an intuitive justification in the fact that communications are often the most expensive operations to be performed in a distributed system. However, no performance measure is available for a language without internal actions, thus equating, e.g., $a|b$ and $a.b + b.a$ like in any other ordinary interleaving-based theory. Furthermore, it may be debatable to count time passing by counting the precise number of internal steps, as if they have all the same duration.

In [20] a "faster than" relation has been studied for a language in the so-called abstract time approach: actions are atomic and time passes "in between" them via explicit global synchronizations. The observer is, unlike us, interacting with the system, imposing – through the interaction – the "real-time" of occurrence for lazy actions; hence, ill-timed paths are not allowed. Time is *preemptive* in the sense that the passing of time can exclude the execution of urgent actions; moreover, it is argued that no sensible notion of precongruence can be defined in presence of choices done on urgent actions. Hence, the authors develop their preorder only on processes with lazy actions. Even if defined in another context and with very different technical assumptions, the basic intuition is similar to ours: E_1 and E_2 are related if they are bisimilar (functionally equivalent) and E_1 performs more quickly than E_2. In a subsequent paper [21], the theory for processes with lazy actions is equipped with an efficiency preorder based on a timed simulation relation, which permits to compare processes not functionally equivalent (e.g., in their approach, the slowest process is nil). Moreover, after showing how to translate Milner's imperative language (see Chapter 8 of [18]) into their language, they show that the parallel composition of two processes can simulate their sequential composition. This result is in the spirit of our theorem showing that the more distributed a process is, the more efficient it is. A direct comparison of our approach with theirs is not so easy, because of the several different assumptions made in the two approaches, and is left for future research.

Outside the process algebra community, we would like to mention the work by W. Vogler [26, 27], that –like ours – is based on a *non-interleaving* framework. He studies a testing-based semantics for timed Petri nets where the duration of transitions may vary from execution to execution and is actually fixed by the tester. Also in this case, further work is needed to understand the relationship between the two approaches.

8.2 Communicating Processes

Extending our theory to communicating processes is easy, in principle: one needs to introduce a new (rule for the) parallel operator (accounting for synchronization). In the setting of durational actions, two synchronization rules have been proposed (one in [14] and the other in [3]), which are different for both the synchronization discipline (the former based on a CSP-like synchronization discipline, the latter on a CCS-like one) and, more importantly, for the timing assumptions. As we will see, both options can be easily accomodated in the present theory, but a lot of the results we have proved in the previous sections do not hold any more.

The intuition behind the rule in [14] is that two processes can synchronize when they perform communicating actions at the same time; if one of the two is able to execute such an action before the other one, then a form of "busy waiting" is allowed. The major drawback of this approach is that performance equivalence is not preserved by parallel composition operator with synchronization. So, it is not a surprise to observe that also performance preorder is not a precongruence: The same counter-example for performance equivalence – reported in [14] – is also a counter-example for the performance preorder. Here, however, we show another, a bit simpler example. Consider the two processes E_1 and E_2 below, already presented in the proof of Proposition 4.5:

$$E_1 \;=\; a.a \,|\, c \;+\; c.a.a \quad \text{ and } \quad E_2 \;=\; a.a \,|\, c \;+\; c.a.a \;+\; c.(a \,|\, a)$$

We know that $E_1 \simeq_p^f E_2$ if and only if $f(a) \leq f(c)$. Hence, let us take a duration function f such that $f(a) = f(c) = 1$. If the parallel operator $|$ allows communication according to the CCS discipline, then the required context is $F = \bar{c}.(\bar{a} \,|\, \bar{a})$, where the duration of the co-actions $\bar{a}$ and $\bar{c}$ is 1 as well. Indeed, it is not difficult to see that $E_1|F \not\sqsubseteq_p^f E_2|F$: it is enough to consider, for instance, the computation where the synchronizations take place between the third addend of E_2 and F; after the synchronization with c and with the first a, we reach a state where an a can be performed at time 2; whatever choice we take, in any state reachable from $E_1|F$ via two synchronizations, a can be performed only at time 3. It is very easy to imagine how this example can be modified to match the CSP synchronization discipline: the occurrences of operator $|$ inside E_1 and E_2 are to be replaced by $||_\emptyset$, F becomes $c.(a \,||_\emptyset\, a)$, while the composition of the two is via $||_{\{a,c\}}$, meaning that the synchronization is forced on actions a and c. Note that, in this case, all actions are visible.

The rule in [2, 3] is stricter, because it prevents any form of "busy waiting": Two processes can synchronize if they are ready to do so exactly at the same time instant. The resulting equivalence is a congruence for parallel composition with synchronization, but it is uncomparable with any of the untimed semantics, e.g. strong bisimulation equivalence, because a lot of potential synchronizations are forbidden. The pruning of some (potential, untimed) transitions may cause some problems, as the following example shows. Assume that $f(\bar{\alpha}) = f(\alpha)$ for all $\alpha \in Act$. Then, $a \mid b \sqsubseteq_p a.b + b.a$ and $\bar{a} \mid \bar{b} \sqsubseteq_p \bar{a} \mid \bar{b}$ but $(a \mid b) \mid (\bar{a} \mid \bar{b})$ is not $\sqsubseteq_p$-related to $(a.b + b.a) \mid (\bar{a} \mid \bar{b})$. In fact, from the state reached by the following transition

$$0 \Rightarrow (a \mid b) \mid (\bar{a} \mid \bar{b}) \stackrel{<\tau, f(a)>}{\longrightarrow} (f(a) \Rightarrow nil \mid 0 \Rightarrow b) \mid (f(a) \Rightarrow nil \mid 0 \Rightarrow \bar{b})$$

a synchronization on b is derivable, while the target state of transition

$$0 \Rightarrow (a.b + b.a) \mid (\bar{a} \mid \bar{b}) \stackrel{<\tau, f(a)>}{\longrightarrow} (f(a) \Rightarrow b) \mid (f(a) \Rightarrow nil \mid 0 \Rightarrow \bar{b})$$

cannot perform a τ. Hence, also this restrictive rule does not originate a precongruence. Again, it is very easy to modify this example to match a CSP synchronization discipline.

As we have seen above, when a CCS-like synchronization discipline is adopted, an invisible action τ is needed (whose duration depends on the duration of the synchronizing actions) to label the transitions of synchronization. Correspondingly, at the syntax level, a delay operator wait n can be also introduced, meaning that only the passage of n time units is observed in the transition. It is not difficult to see that, if we add the delay operator to our communication free language, we still get congruence results for the strong (i.e., τ-sensitive) semantics studied in this paper, characterized by axioms in the usual way.

Moreover, to force synchronizations, a restriction operator is also added to CCS. In a full language with communication, delay and restriction operators, for the strong semantics studied in this paper, another instructive example of lack of congruence results for parallel composition can be given. Consider the two processes P and Q below:

$$P = (c.\text{wait } 3.a.b | c.\text{wait } 3.a.b | \bar{a}) \backslash \{a\}$$

$$Q = (c.\text{wait } 3.a.b | c.(\text{wait } 3.a.b + a.\text{wait } 3.b) | \bar{a}) \backslash \{a\}$$

and assume that the duration of actions a and c is 3, as well as of their co-actions $\bar{a}$ and $\bar{c}$. It is easy to see that $P \simeq_p Q$, because the additional behaviour of Q (synchronization on a before the wait of 3 time units) is observationally indistinguishable: in any case b will be executed starting from 6 time units after the completion of c. Now, consider $R = P | \bar{c}.\bar{c}$ and $S = Q | \bar{c}.\bar{c}$. It is not difficult to see that $S \not\sqsubseteq_p R$. In fact, after the synchronization on the leftmost c in Q, the subsequent synchronization with the rightmost c in Q and then the synchronization on a from the subterm $a.$wait $3.b$ in Q, we reach a state of S which can execute b only at time 15 after one τ-labelled transition completed at time 12. On the contrary, in any reachable state from R via three τ transitions labelled by 3, 6 and 9 time units respectively, it is always possible to have b's with completion time 12.

Furthermore, it is no longer true that the location preorder implies the performance preorder. Consider processes $P = (a.b | \bar{b}.c) \backslash \{b\}$ and $Q = (a.d | \bar{d}.c) \backslash \{d\}$ and assume that $f(b) > f(d)$. Then $P \sqsubseteq_l Q$ while $P \not\sqsubseteq_p Q$. Indeed, this example shows that the origin of the failure of the property above is that action τ has no predetermined duration (action duration functions are not defined on τ's), i.e., its duration may be different in different occurrences. This example also shows the failure of another property: causal equivalence [10] implies performance equivalence, as proved in [14] for a language without silent moves.

As the preorder in [4] is based on counting internal actions, we want to show the obvious fact that the performance preoder on the extended language with communication is incomparable with that efficiency preorder. Consider the two processes $P = (a.b.d | \bar{b}.\bar{d}.c) \backslash \{b, d\}$ and $Q = (a.e | \bar{e}.c) \backslash \{e\}$. It is clear that Q is more efficient than P in the approach of [4], but it is also clear that $P \sqsubseteq_p Q$ if $f(b) + f(d) < f(e)$. Indeed, it is part of our everyday experience that a direct communication can be slower than another passing through a larger number of intermediate routers, if the direct line has very small bandwidth. So we claim that our preorder is, in this respect, more realistic.

8.3 Open Problems and Future Research

Here we briefly recall some open problems, partly already hinted in previous sections, as lines for further research.

Congruence

As we are convinced that the communication rule of [14] is more realistic in general, further work will be devoted to single out the coarsest precongruence contained in the performance preorder with that communication rule. Moreover, we plan to study if there are other sensible synchronization rules for which performance preorder is a precongruence.

Decomposition

Another, quite interesting line of research is the study of "optimal implementations" for a given specification: given a specification E_1, find the fastest E_2 which is functionally equivalent to E_1. This means that we would like to study algorithms for finding the minimal elements in the performance preorder. Due to the comparison with the location preorder, it seems clear that this problem is tightly related to the problem of finding *parallel decompositions* of processes, where the notion of decomposition we have in mind has some connection with the one in [19]. Preliminary results in this directions are reported in [9].

Concise Representation and Decidability Results

The model we have used to define performance preorder, even if appealing because simple and intuitive, suffers of the same problems of all the non-interleaving equivalences associating dynamically informations to the states of their underlying transition systems (i.e., location, causal or ST equivalences). In particular, the transition systems associated to recursive agents are never finite states, as no cycles are created by the operational semantics. Indeed, in our case local clocks, associated to the sequential components of a process, increase as the computation proceeds, hence preventing the creation of any loop. For this kind of problems, decidability results and proof systems for these non-interleaving semantics are possible only if one is able to find some alternative finite characterization by using, e.g., symbolic techniques. Indeed, in the literature, this is what happened, for instance, for location equivalence: originally proposed in [6] as an infinite states transition system, it was then characterized finitely by, e.g., [7, 8]. By analogy, we hope that it is possibile to find a solution for our preorder.

Ill-timed

The ill-timed phenomenon, present in our approach, is sometimes criticized because some researchers consider impossible to accept that there could be a distinction between the time when an action is seen by an observer (the order of generation of the transition in the computation tree for a process) and the actual time when such an action has been executed (the clock value appearing in the transition labels). We strongly defend this point of view for the following reasons:

1. In a distributed environment it is difficult to think of a global observer being able to see all the executed actions "synchronously". The best (s)he can do is to collect the observations that each observer in each site is able to see; this collection phase corresponds to building the computation tree for the process, where it is no more required to have an agreement between the generation time and the execution time.

2. In absence of ill-timed paths, performance preorder does no longer relate functionally equivalent processes, at least if the functional equivalence is based on bisimulation. Indeed, if

$f(a) > f(b)$, then $a|b$ and $a.b + b.a$ cannot be related as the observer can only see the execution of b first and then the execution of a from $a|b$, while in $a.b + b.a$ both sequences are possible. It is clear that one way out is to change the choice of the underlying functional equivalence to something asymmetric, such as a simulation relation. This is, indeed, the way taken in [21].

3. From a technical point of view, it is quite unfortunate to cut the ill-timed paths. Apart from the difficulties of defining a structural semantics ensuring this requirement (see, e.g., [24]), we want to stress that the game is not worth-playing. In [11] it is defined performance equivalence on well-timed paths only, called *increasing* equivalence. There it is observed that increasing equivalence is less discriminating than performance equivalence. In particular, terms $a.c|b$ and $a|b.c$ become increasing equivalent, if $f(a) = f(b)$. Moreover, it is observed that increasing equivalence is not a congruence for parallel composition with synchronization a là [14]. A. Rensink in a private communication observed (the quite surprising fact) that the coarsest congruence for increasing equivalence – whatever it is – must have also the ill-timed paths. We are presently investigating all the consequences of taking this point of view. Some preliminary results are reported in [11].

Acknowledgements

The first author thanks Rocco De Nicola for his encouragement to explore this area of research. The second author wishes to thank Walter Vogler for helpful comments and suggestions.

References

1. L.Aceto, *On Relating Concurrency and Nondeterminism*, Technical Report SI/RR–89/06, University of Sussex, 1989.

2. L.Aceto, D.Murphy, *On the Ill–Timed but Well–Caused*, In CONCUR'93 (E.Best, Ed.), Springer-Verlag LNCS **715**, pp.97-111, 1993.

3. L.Aceto, D.Murphy, *Timing and Causality in Process Algebra*, Acta Informatica, 33(4), June 1996.

4. S.Arun-Kumar and M.Hennessy, *An Efficiency Preorder for Processes*, Acta Informatica **29**, pp.737-760, 1992.

5. S.Arun-Kumar, V.Natarajan, *Conformance: A Precongruence Close to Bisimilarity*, in Proc. *Structures in Concurrency* (STRICT'95), Springer Workshops in Computing Series, 1995.

6. G.Boudol, I.Castellani, M.Hennessy, A.Kiehn, *A Theory of Processes with Localities*, Formal Aspects of Computing **6**, pp. 165-200, 1993.

7. I.Castellani, *Observing Distribution in Processe: Static and Dynamic Localities*, INRIA RR n. 2276, 1994.

8. F.Corradini, R.De Nicola, *Locality Based Semantics for Process Algebras*, Acta Informatica, to appear.

9. F.Corradini, R.Gorrieri, D.Marchignoli, *On the Parallelization of Concurrent Systems I: Basic Definitions and Properties*, Technical Report of Dept. of Computer Science, University of Bologna, July 1996, to appear.

10. Ph.Darondeau, P.Degano, *Causal trees*, In Automata, Languages and Programming, Springer-Verlag LNCS **372**, 1989.

11. D. Della Godenza, *Equivalenze osservazionali per sistemi temporizzati*, Master thesis, University of Bologna, November 1995.

12. G-L.Ferrari and U.Montanari, *Dynamic Matrices and the Cost Analysis of Concurrent Systems*, in Proc. *AMAST'95*, 1995.

13. R.Gorrieri, C.Laneve, *Split and ST Bisimulation Semantics*, Information and Computation, **118** (2), 1995, pp. 272-288.

14. R.Gorrieri, M.Roccetti and E. Stancampiano, *A Theory of Processes with Durational Actions*, Theoretical Computer Science **140** (1), March 1995, pp. 73-94.

15. R.J. van Glabbeek, F.Vaandrager, *Petri Net Models for Algebraic Theories of Concurrency*, In the Proceedings of PARLE II, Springer-Verlag LNCS **259**, pp. 224-242, 1987.

16. J.F.Groote, F.Vaandrager, *Structured Operational Semantics and Bisimulation as a Congruence*, Information and Computation **100** (2), pp. 202-260, 1992.

17. M. Hennessy and R.Milner, *Algebraic Laws for Nondeterminism and Concurrency*, Journal of the ACM **32**(1), pp.137-161, 1985.

18. R.Milner, *Communication and concurrency*, International series on computer science, Prentice Hall International, 1989.

19. R.Milner, F.Moller, *Unique Decomposition of Processes.* Theoretical Computer Science **107**, pp. 357-363, 1993.

20. F.Moller, C.Tofts, *Relating Processes with Respect to Speed.* In CONCUR'91, LNCS **527**, Springer-Verlag, pp. 424-438, 1991.

21. F.Moller, C.Tofts, *Reasoning about resource Bounded Computations in Real- Timed Process Calculi.* In Proc. SoftSem'93, 1993.

22. V.Natarajan, R. Cleaveland, *An Algebraic Theory of Process Efficiency*, In Proc. *11th Symp. on Logic in Computer Science*, IEEE CS Press, 1996, to appear.

23. M.Nielsen, P.S.Thiagarajan, *Degrees of Nondeterminism and Concurrency*, In Proceedings 4th Conf.on Found.of Soft.Tech.and Theor.Computer Science, Springer-Verlag, LNCS **181**, pp. 89-117, 1984.

24. J. Quemada, D. Frutos, A. Azcorra, *TIC: A TImed Calculus*, Formal Aspects of Computing **5**, pp. 224-252, 1993.

25. E. Stancampiano, *La Teoria della Performance Bisimulazione*, Master Thesis, Università di Bologna, 1993.

26. W. Vogler, *Timed Testing of Concurrent Systems*, Information and Computation **121**(2), pp. 149-171, 1995.

27. W. Vogler, *Faster Asynchronous Systems*, In Proc. CONCUR'95, LNCS 962, pp. 299-312, Springer-Verlag, 1995.

9 Appendix: The Location Preorder

In [6] a preorder is introduced which relates processes with respect to their degree of spatial distribution. For E_1 and E_2 processes, $E_1 \sqsubseteq_l E_2$, will stand for the fact that E_1 and E_2 are bisimilar but E_1 is observably more distributed than E_2. This is reached by introducing a new transition system, called location transition system, that relates pairs of processes and sequences of locations. The sequences of locations provide indication on where actions have been executed. The new transition system is of the form

$$(R, Act, Loc, \{\xrightarrow{a,u} \mid a \in Act, u \in Loc^*\})$$

where Loc is an infinite set of atomic location names ranged over l_1, l_2,.... R is the set of processes with locations obtained by extending the grammar generating P with a new operator, called prefix operator and denoted by $l :: \_$ with $l \in Loc$. This operator prefixes an agent E with a location name l so that $l :: E$ means that agent E resides at location l. In this way it is possible to keep track of the observed locations.

Let R be the set of located terms genered by the following grammar:

$$r \ ::= nil \ \Big| \ a.E \ \Big| \ E + E \ \Big| \ r \mid r \ \Big| \ l :: r \ \Big| \ \mathrm{rec}\ x.\, E$$

where $l \in Loc$ and $a.E$, rec $x.\,E$ are closed P processes.

The structural rules for the operational semantics of located states are listed in Table 8.

$LT1$	$\dfrac{}{a.E \xrightarrow{a,l} l :: E}$	$l \in Loc$	
$LT2$	$\dfrac{r \xrightarrow{a,u} r'}{k :: r \xrightarrow{a,ku} k :: r'}$		
$LT3$	$\dfrac{E_1 \xrightarrow{a,u} r_1}{E_1 + E_2 \xrightarrow{a,u} r_1}$	$LT4$	$\dfrac{E_2 \xrightarrow{a,u} r_2}{E_1 + E_2 \xrightarrow{a,u} r_2}$
$LT5$	$\dfrac{r \xrightarrow{a,u} r'}{r \mid s \xrightarrow{a,u} r' \mid s}$	$LT6$	$\dfrac{s \xrightarrow{a,u} s'}{r \mid s \xrightarrow{a,u} r \mid s'}$
$LT7$	$\dfrac{E[\mathrm{rec}\ x.\,E/x] \xrightarrow{a,u} r'}{\mathrm{rec}\ x.\,E \xrightarrow{a,u} r'}$		

Table 8: Location Transition Relation.

We now give the formal definition of location preorder. It is based on a preorder over sequences of locations. For u, $v \in Loc^*$, $u \leq v$, denotes that u is a subword of v, i.e. $u = l_1 l_2 ... l_n$, where $l_i \in Loc$ and there exists $v_1 ... v_{n+1} \in Loc^*$ such that $v_1 l_1 v_2 l_2 ... v_n l_n v_{n+1} = v$. Let $r \in R$, $loc(r)$ will denote the set of atomic location names occurring in r.

Definition 9.1 (*Locality Preorder*)

1) Let Rel denote the set of binary relations over R. The functional $L : Rel \rightarrow Rel$ is defined, for each $\Re \in Rel$, as follows: $(r_1, r_2) \in L(\Re)$, if for each $a \in Act$,

   i) $r_1 \xrightarrow{a,u} r_1'$ with $u = l_1...l_k l$ and $l \notin (loc(r_1) \cup loc(r_2))$, implies $r_2 \xrightarrow{a,v} r_2'$ with $v = l_1'...l_m' l$, such that $u \leq v$ and $(r_1', r_2') \in \Re$;

   ii) $r_2 \xrightarrow{a,u} r_2'$ with $u = l_1...l_k l$ and $l \notin (loc(r_1) \cup loc(r_2))$, implies $r_1 \xrightarrow{a,v} r_2'$ with $v = l_1'...l_m' l$, such that $v \leq u$ and $(r_1', r_2') \in \Re$.

2) A relation $\Re \in Rel$ will be called a $L - bisimulation$ if $\Re \subseteq L(\Re)$.

3) Let E_1, $E_2 \in P$ then $E_1 \sqsubseteq_l E_2$ iff there exists a $L - bisimulation$ $\Re$ such that $(E_1, E_2) \in \Re$.

The original definition in [6] does not require location name l in clauses (i) and (ii) to be new. However, the one presented above is equivalent to the original definition and permits us to easily compare location preorder with performance preorder. Due to this in the following we restrict the attention on located states r with distinct atomic location names occurring in it.

The following of this section is entirely devoted to relate location and performance preorder. We are interested in the comparison of P terms so that we can simplify the syntax of extended states by replacing production $t = t_1 + t_2$ (useful for the axiomatization of performance preorder) with $t = E_1 + E_2$ where E_1 and E_2 are P terms. We denote with $\mathbf{T}_l$ the set of terms genered by the new grammar. This will permits us to give a more direct comparison.

The following proposition shows that terms obtained by applying rules in Table 3 to $\mathbf{T}_l$ terms, give rise to terms that are in $\mathbf{T}_l$ again. Thus, since P is a proper subset of $\mathbf{T}_l$, every sequence of application of rules in Table 3 give a target state which is in the syntax. In other words the operational semantics never gives rise to terms of the form $(n_1 \Rightarrow E_1) + (n_2 \Rightarrow E_2)$, with $n_1 \neq n_2$.

Proposition 9.1 If $t \in \mathbf{T}_l$ and $t \xrightarrow{\langle a,n \rangle} t'$ then $t' \in \mathbf{T}_l$.

Proof: By induction on the depth of the proof of derivation $t \xrightarrow{\langle a,n \rangle} t'$. $\square$

Given a location state $r \in R$, we can obtain an extended state t by substituting every location prefix $l ::$ _ occurring in r with a timing prefix $n \Rightarrow$ _. To this, some new notation is needed. $[\tilde{l}/\tilde{n}]$ with $\tilde{l} = (l_1, ..., l_k)$, $l_i \in Loc$ and $\tilde{n} = (n_1, ..., n_k)$, $n_i \in N^+$, will be used to abbreviate multiple substitutions $[l_1/n_1, ..., l_k/n_k]$ in a located term r: $r[\tilde{l}/\tilde{n}]$ with $loc(r) \subseteq \{l_1, ..., l_k\}$ will denote the extended term obtained from r by substituting n_i for the instances of location name l_i appearing in r, for each $l_i/n_i \in [l_1/n_1, ..., l_k/n_k]$. Formally $r[\tilde{l}/\tilde{n}]$ with $\tilde{l} = (l_1, ..., l_k)$ and $loc(r) \subseteq \{l_1, ..., l_k\}$, denotes the unique extended term obtained by applying axioms in Table 9 as rewrite rules from left to right.

Propositions below relate located and timed operational semantics. We show that for each transition of the extending state $r[\tilde{l}/\tilde{n}]$ there exists a corresponding transition of the located state r and vice versa.

Proposition 9.2 Let r be a located state and t be an extended state such that $t = r[\tilde{l}/\tilde{n}]$ for some $\tilde{l} = (l_1, ..., l_k)$, $\tilde{n} = (n_1, ..., n_k)$ and $loc(r) \subseteq \{l_1, ..., l_k\}$.
If $t \xrightarrow{\langle a,n \rangle} t'$ then $\forall l \in Loc$ and $l \notin \{l_1, ..., l_k\}$, $r \xrightarrow{a,u} r'$ with $u = l_1'...l_m' l$, $t' = r'[\tilde{l}/\tilde{n}, l/f(a)]$ and $n = n_1' + ... + n_m' + f(a)$ where $l_i'/n_i' \in \tilde{l}/\tilde{n}$ for each $i \in [1..m]$.

Proof: By induction on the depth of the proof of $t \xrightarrow{\langle a,n \rangle} t'$. We distinguish some cases depending on the syntactic structure of r.

(1)	$nil[l_1/n_1, ..., l_k/n_k]$	$=$	nil
(2)	$a.E[l_1/n_1, ..., l_k/n_k]$	$=$	$a.E$
(3)	$(E_1 + E_2)[l_1/n_1, ..., l_k/n_k]$	$=$	$E_1 + E_2$
(4)	$\mathrm{rec}\ x.E[l_1/n_1, ..., l_k/n_k]$	$=$	$\mathrm{rec}\ x.E$
(5)	$(r_1 \vert r_2)[l_1/n_1, ..., l_k/n_k]$	$=$	$(r_1[l_1/n_1, ..., l_k/n_k]) \vert (r_2[l_1/n_1, ..., l_k/n_k])$
(6)	$(l_i :: r)[l_1/n_1, ..., l_i/n_i, ..., l_k/n_k]$	$=$	$n_i \Rightarrow (r[l_1/n_1, ..., l_i/n_i, ..., l_k/n_k])$

Table 9: Multiple Substitutions.

1. $r = a.E$. Then $t = a.E[\tilde{l}/\tilde{n}] = a.E$ and $a.E \xrightarrow{a,l} l :: E$ for each $l \in Loc$, $l \notin \{l_1, ..., l_k\}$. Moreover $(l :: E)[\tilde{l}/\tilde{n}, l/f(a)] = f(a) \Rightarrow (E[\tilde{l}/\tilde{n}, l/f(a)]) = (f(a) \Rightarrow E)$ because E is a P agent.

2. $r = E_1 + E_2$. Then $t = (E_1 + E_2)[\tilde{l}/\tilde{n}] = E_1 + E_2$. For $E_1 + E_2 \xrightarrow{\langle a,n \rangle} t'$ we have to distinguish two cases:

   a. $E_1 \xrightarrow{\langle a,n \rangle} t'$ and by induction hypothesis $E_1 \xrightarrow{a,ul} r'$ with $t' = r'[\tilde{l}/\tilde{n}, l/f(a)]$ and $n = n'_1 + ... + n'_k + f(a)$ if $u = l'_1 ... l'_k$ and $l'_i/n'_i \in \tilde{l}/\tilde{n}$. By $LT3$ it follows also that $E_1 + E_2 \xrightarrow{a,ul} r'$.

   b. $E_2 \xrightarrow{\langle a,n \rangle} t'$. This case is symmetrical to the previous one.

3. $r = \mathrm{rec}\ x.\ E$. Then $t = (\mathrm{rec}\ x.\ E)[\tilde{l}/\tilde{n}] = \mathrm{rec}\ x.\ E$ and $\mathrm{rec}\ x.\ E \xrightarrow{\langle a,n \rangle} t'$ if $E[\mathrm{rec}\ x.\ E/x] \xrightarrow{\langle a,n \rangle} t'$. By induction hypothesis we can assume a transition $E[\mathrm{rec}\ x.\ E/x] \xrightarrow{a,ul} r'$ for each $l \in Loc$, $l \notin \{l_1, ..., l_k\}$ with $t' = r'[\tilde{l}/\tilde{n}, l/f(a)]$ and $n = n'_1 + ... + n'_k + f(a)$ if $u = l'_1 ... l'_k$ and $l'_i/n'_i \in \tilde{l}/\tilde{n}$. By $LT7$, $\mathrm{rec}\ x.\ E \xrightarrow{a,ul} r'$ follows.

4. $r = r'_1 | r'_2$. Then $t = (r'_1 | r'_2)[\tilde{l}/\tilde{n}] = (r'_1[\tilde{l}/\tilde{n}]) | (r'_2[\tilde{l}/\tilde{n}]) = t'_1 | t'_2$. For $t \xrightarrow{\langle a,n \rangle} t'$ we distinguish two cases:

   a. $t'_1 | t'_2 \xrightarrow{\langle a,n \rangle} t''_1 | t'_2$ if $t'_1 \xrightarrow{\langle a,n \rangle} t''_1$. By induction hypothesis since $t'_1 = r'_1[\tilde{l}/\tilde{n}]$ follows $r'_1 \xrightarrow{a,ul} r''_1$ for each $l \in Loc$, $l \notin \{l_1, ..., l_k\}$ and $n = n'_1 + ... + n'_k + f(a)$ if $u = l'_1 ... l'_k$ and $l'_i/n'_i \in \tilde{l}/\tilde{n}$. Moreover $t''_1 = r''_1[\tilde{l}/\tilde{n}, l/f(a)]$. By $LT5$, $r'_1 | r'_2 \xrightarrow{a,ul} r''_1 | r'_2$ and $t''_1 | t'_2 = (r''_1 | r'_2)[\tilde{l}/\tilde{n}, l/f(a)]$ because $(r''_1 | r'_2)[\tilde{l}/\tilde{n}, l/f(a)] = (r''_1[\tilde{l}/\tilde{n}, l/f(a)]) | (r'_2[\tilde{l}/\tilde{n}, l/f(a)])$ and $t''_1 = r''_1[\tilde{l}/\tilde{n}, l/f(a)]$ by induction hypothesis and $t'_2 = r'_1[\tilde{l}/\tilde{n}, l/f(a)]$ because $t'_2 = r'_1[\tilde{l}/\tilde{n}]$ and $l \notin loc(r'_1)$.

   b. $t'_1 | t'_2 \xrightarrow{\langle a,n \rangle} t'_1 | t''_2$, this case is symmetric to the previous one.

5. $r = l_i :: r'$. Then $t = (l_i :: r')[\tilde{l}/\tilde{n}] = n_i \Rightarrow (r'[\tilde{l}/\tilde{n}]) = n_i \Rightarrow t'$ with $l_i/n_i \in \tilde{l}/\tilde{n}$ and $t' = r'[\tilde{l}/\tilde{n}]$. Suppose $(n_i \Rightarrow t') \xrightarrow{\langle a,n+n_i \rangle} (n_i \Rightarrow t'_1)$ if $t' \xrightarrow{\langle a,n \rangle} t'_1$. By induction hypothesis $r' \xrightarrow{a,ul} r'_1$ with $l \in Loc$, $l \notin \{l_1, ..., l_k\}$ and $n = n'_1 + ... + n'_k + f(a)$ if $u = l'_1 ... l'_k$ where

$l'_i/n'_i \in \tilde{l}/\tilde{n}$. Moreover $t'_1 = r'_1[\tilde{l}/\tilde{n}, l/f(a)]$. By $LT2$ follows $l_i :: r' \xrightarrow{a,l_iul} l_i :: r'_1$ and $n_i \Rightarrow t'_1 = (l_i :: r'_1)[\tilde{l}/\tilde{n}, l/f(a)]$ because $(l_i :: r'_1)[\tilde{l}/\tilde{n}, l/f(a)] = n_i \Rightarrow (r'_1[\tilde{l}/\tilde{n}, l/f(a)])$ and by induction hypothesis $r'_1[\tilde{l}/\tilde{n}, l/f(a)] = t'_1$.

□

The reverse of the Proposition above is stated below and can be proved following a similar inductive proof on the depth of the proof of transitions.

Proposition 9.3 Let r be a located state and t be an extended state such that $t = r[\tilde{l}/\tilde{n}]$ for some $\tilde{l} = (l_1, ..., l_k)$, $\tilde{n} = (n_1, ..., n_k)$ and $loc(r) \subseteq \{l_1, ..., l_k\}$.
If $r \xrightarrow{a,u} r'$ with $u = l'_1...l'_m l$, $l \in Loc$ and $l \notin \{l_1, ..., l_k\}$, then $t \xrightarrow{\langle a,n \rangle} t'$ with $n = n'_1 + ... + n'_m + f(a)$ and $l'_i/n'_i \in \tilde{l}/\tilde{n}$ for each $i \in [1..m]$.

We are now ready to prove the case (i) of Proposition 6.3.

Proposition 6.3 (i). For each E_1, E_2 agents, $E_1 \sqsubseteq_l E_2$ implies $E_1 \sqsubseteq_p^i E_2$.
Proof: If $E_1 \sqsubseteq_l E_2$ then there exists a $L-bisimulation$ $\Re$ such that $(E_1, E_2) \in \Re$. We show that relation:

$$\begin{array}{rl} \Re' = \{(t_1, t_2) \mid & t_1 = r_1[\tilde{l}/\tilde{n}],\ t_2 = r_2[\tilde{l}/\tilde{n}],\ (r_1, r_2) \in \Re \text{ for some } \tilde{l},\ \tilde{n} \text{ such that} \\ & \tilde{l} = (l_1, ..., l_k),\ \{l_1, ...l_k\} = loc(r_1) \cup loc(r_2) \text{ and } \tilde{n} = (n_1, ..., n_k)\} \end{array}$$

is a $G-bisimulation$ for (E_1, E_2). To this consider a generic pair $(t_1, t_2) \in \Re'$ and suppose:

1. $t_1 \xrightarrow{\langle a,n \rangle} t'_1$. Since $(t_1, t_2) \in \Re'$ then there exists a pair $(r_1, r_2) \in \Re$ such that $t_1 = r_1[\tilde{l}/\tilde{n}]$, $t_2 = r_2[\tilde{l}/\tilde{n}]$, $\tilde{l} = (l_1, ..., l_k)$, $\{l_1, ...l_k\} = loc(r_1) \cup loc(r_2)$ and $\tilde{n} = (n_1, ..., n_k)$. By Proposition 9.2, it follows that $r_1 \xrightarrow{a,u} r'_1$ with $u = l'_1...l'_k l$, $l \in Loc$, $l \notin loc(r_1) \cup loc(r_2)$, $t'_1 = r'_1[\tilde{l}/\tilde{n}, l/f(a)]$ and $n = n'_1 + ... + n'_k + f(a)$ for $l'_i/n'_i \in \tilde{l}/\tilde{n}$ for any i.

   Since $(r_1, r_2) \in \Re$, $r_2 \xrightarrow{a,v} r'_2$ with $v = l''_1...l''_n l$, $l \in Loc$, $l \notin loc(r_1) \cup loc(r_2)$, $u \leq v$ and $(r'_1, r'_2) \in \Re$. By Proposition 9.3 follows $t_2 \xrightarrow{a,n'} t'_2$ with $t'_2 = r'_2[\tilde{l}/\tilde{n}, l/f(a)]$ and $n' = n''_1 + ... + n''_n + f(a)$ with $l''_i/n''_i \in \tilde{l}/\tilde{n}$. Since $u \leq v$ then by definition of superword there are w_1,...,$w_k \in Loc^*$ such that $v = w_1 l'_1 w_2 l'_2 ... w_{k-1} l'_k w_k$. Thus $n \leq n'$ and clearly $(t'_1, t'_2) \in \Re'$.

2. $t_2 \xrightarrow{\langle a,m \rangle} t'_2$. This case is similar to the previous one.

□